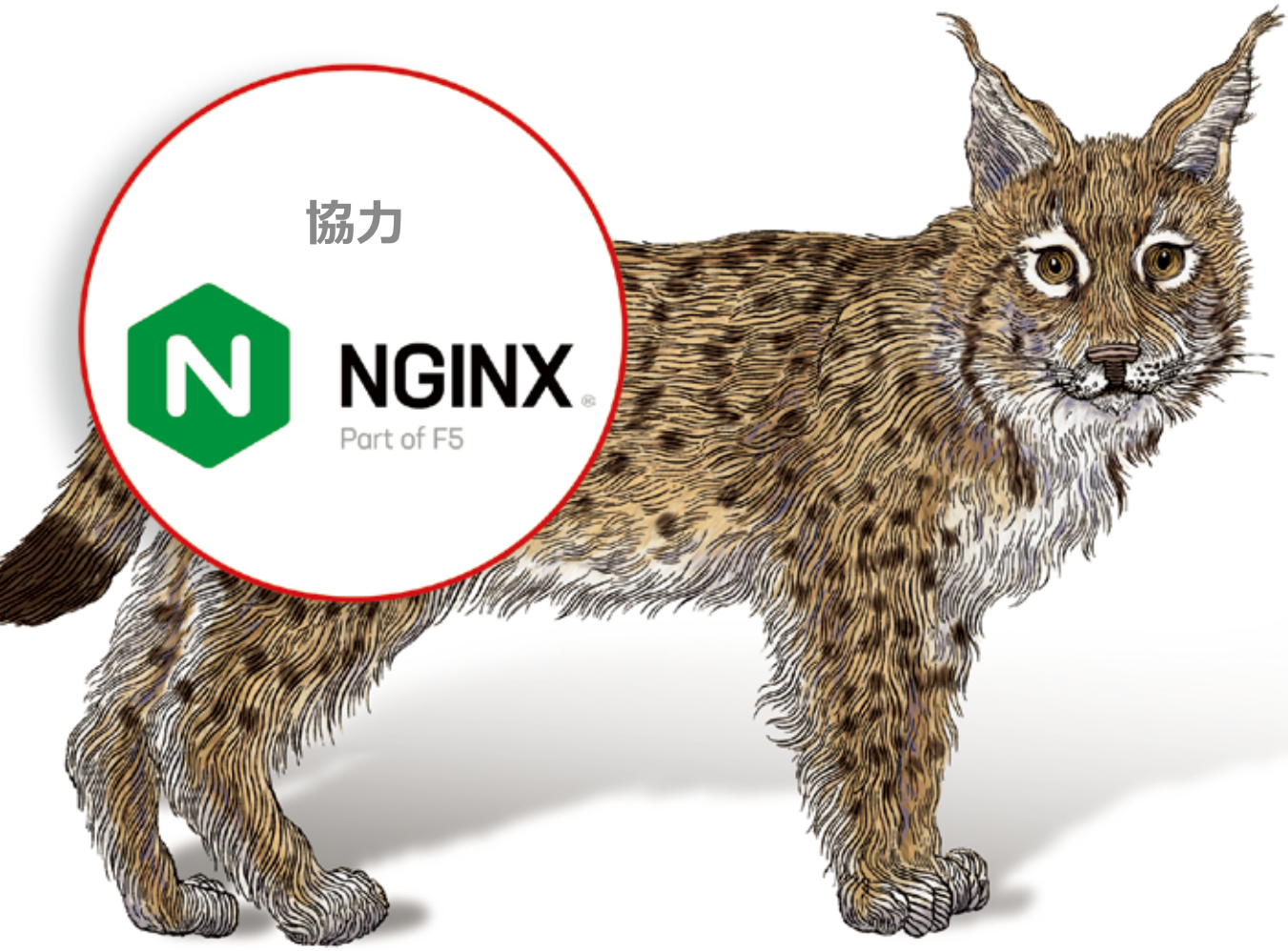
O'REILLY®

第三版

NGINX クックブック

高機能ロードバランシングの設定レシピ



Derek DeJonghe

O'REILLY®

NGINXクックブック

NGINX は、現在最も広く使用されているWebサーバーの1つです。その理由の一つは、HTTPおよびその他のネットワークプロトコルのロードバランサー、そしてリバースプロキシサーバーとしての機能を備える点です。この改訂版クックブックは、アプリケーション配信を行う際に使われる機能や設定の例を解説します。ここで紹介する実用的な設定レシピは、オープンソース版または商用版を使用し、さまざまなユースケースにおける課題とその解決方法を提供します。

n層やマイクロサービスの設計などの最新のWebアーキテクチャ、およびTCPやHTTPなどの一般的なWebプロトコルを使用されているユーザー向けに、この解説書では、NGINXアプリケーション配信プラットフォームでの、セキュリティ設定やソフトウェアロードバランシング、モニタリング、メンテナンスなど、世の中で幅広く活用されているソリューションを説明します。また本書では、無料オープンソース版であるNGINXと、商用版NGINX Plusの両方の高度な機能についても説明します。

以下のレシピをご覧いただけます：

- **HTTP、TCP、UDPの高度なロードバランシング**
- **暗号化されたトラフィック、安全なリンク、HTTP認証サブリクエストなどによるアクセスの保護**
- **NGINXをGoogle、AWS、Azure Cloud Servicesにデプロイする方法**
- **SAML環境におけるサービスプロバイダーとしてのNGINX Plus**
- **HTTP/3（QUIC）、OpenTelemetry、njsモジュール**

Derek DeJongheは、Amazon Web Servicesの認定プロフェッショナルであり、Linux/Unixベースのシステムおよびwebアプリケーションの専門家です。Web開発、システム管理、ネットワーク構築分野における経験があり、クラウドにおける価値ある人材です。インフラストラクチャ管理、構成管理、継続的インテグレーションに注力し、DevOpsツールの開発や、複数のマルチテナント型SaaS製品のシステム、ネットワーク、デプロイの保守にも携わっています。

SYSTEM ADMINISTRATION

Twitter: @oreillymedia
linkedin.com/company/oreilly-media
youtube.com/oreillymedia

F5 NGINX PlusとF5 NGINX App Protectの無料トライアル

マイクロサービスのための高性能なアプリケーション配信およびセキュリティをお試しください

NGINX Plusは、ソフトウェアロードバランサー、APIゲートウェイ、マイクロサービスプロキシです。

NGINX App Protectは、実績のあるF5テクノロジーに基づいて構築され、最新のアプリケーションおよびDevOps環境向けに設計された、軽量かつ高速なWebアプリケーションファイアウォール（WAF）です。

コスト削減

ハードウェアのアプリケーション配信コントローラーやWAFと比較して、大幅なコスト削減を実現します。期待されるすべてのパフォーマンスと機能を提供します。

複雑さの軽減

ロードバランサー、APIゲートウェイ、マイクロサービスプロキシ、Webアプリケーションファイアウォールが揃った唯一のオールインワンソリューションであり、インフラストラクチャ乱立の解消に役立ちます。

エンタープライズ対応

NGINX PlusとNGINX App Protectは、DevOpsやCI/CD環境と統合しながら、セキュリティ、拡張性、耐障害性といったエンタープライズ要件に応えます。

高度なセキュリティ

拡大し続けるサイバー攻撃やセキュリティの脆弱性からアプリとAPIを守ります。

ダウンロード：nginx.com/freetrial

NGINX
Part of F5

第三版

NGINXクックブック

高機能ロードバランシングの設定レシピ

Derek DeJonghe

北京・ボストン・ファーンハム・セヴァストポリ ・東京 O'REILLY®

NGINX Cookbook

Derek DeJonghe著

印刷：アメリカ合衆国

O′Reillyの書籍は、教育、ビジネス、または販売促進目的で購入できます。ほとんどの書籍はオンラインでもご利用いただけます (*http://oreilly.com*)。詳細情報は、企業/教育機関セールス部門までお問合せください。800-998-9938 または*corporate@oreilly.com*.

アクイジションエディター：John Devins
開発エディター：Gary O′Brien
プロダクションエディター：Clare Laylock
コピーエディター：Piper Editorial Consulting, LLC
校正者：Kim Cofer

索引：Potomac Indexing, LLC
内装デザイナー：David Futato
カバーデザイナー：Karen Montgomery
イラストレーター：Kate Dullea

2020年11月：初版
2022年 5 月：第二版
2024年 2 月：第三版

第三版の改訂履歴
2024年1月29日：初版リリース

リリースの詳細情報につきましては、こちらをご覧ください：
http://oreilly.com/catalog/errata.csp?isbn=9781098158439

O′ReillyのロゴはO′Reilly Media, Inc.の登録商標です。NGINXクックブックの表紙の画像、および関連するトレードドレスは、O′Reilly Media、Inc.の商標です。

CK20221109_J

目次

まえがき

NGINXクックブックの2024年版へようこそ。O'Reilly はNGINXクックブックを9年間出版してきましたが、その間、NGINXに定期的に導入される多くの改良を反映させるため、内容の更新を続けています。NGINXは現在、世界で最も人気のあるWebサーバーです。NGINXは、最初のバージョンがリリースされた2004年以降、進化を続け、最新のアプリケーションのスケーリング、セキュリティ、配信に携わる人々のニーズに応えています。柔軟性と拡張性を重視して設計されたNGINXは、Webサーバーだけでなく、ロードバランシング、リバースプロキシ、APIゲートウェイにまで機能を拡張できました。NGINXの価値は、主要なパブリッククラウドやCDNが提供するロードバランシングサービスの多くがNGINXのコードに基づいていることからも証明されています。

NGINXは、新しい領域への拡張と重要な能力の追加も続けています。NGINXとネイティブに統合されているNGINX Ingress Controller for Kubernetesは、Kubernetesの世界で重要な要件である東西トラフィックと南北トラフィックの両方を管理するための主要な機能を提供します。また、NGINXは、認証とセキュリティの機能を一貫して拡張してきました。これが成り立つのは、NGINXが、現代の分散アプリケーションの基本要件であるスピード、レジリエンス、アジリティを提供し続けているからこそです。

NGINXクックブックは、NGINXを最もよく知る人々による、頼りになるNGINXガイドです。小規模なプロジェクトでNGINXオープンソースを実行する場合でも、複数の地域にまたがる企業デプロイメントでNGINX Plusを実行する場合でも、あるいはローカルまたはクラウドで実行する場合でも、NGINXクックブックはNGINXを最大限に活用する上で役立ちます。わかりやすい100を超えるレシピを集めたこのクックブックでは、NGINXのインストール方法、さまざまなユースケースに応じた設定方法、セキュリティやスケールの方法、一般的な問題のトラブルシューティングの方法を解説します。

2024年版では、NGINXの新機能を反映するために多くのセクションが更新され、拡張されたセキュリティ機能やHTTP/2、HTTP/3（QUIC）、gRPCの効果的な活用に焦点を当てた全く新しいセクションが追加されています。急速に変化するテクノロジーとアプリケーションの世界に合わせ、NGINXも変化し、皆様の成功に貢献し続けています。

今日のNGINXのすべては、拡張性、信頼性、高速性、安全性に優れたWebサービスを提供するという当初のビジョンに基づき、私たちのコミュニティがどんな環境でも、どんな規模でも、必要なアプリを自信と信頼をもって保護およびデプロイできるように設計されています。

—Peter Beardmore

F5 NGINXプロダクトマーケティング部長

序文

*NGINX クックブック*は、実際に直面するであろうアプリケーション配信の課題に対しわかりやすい解決策を提供することを目的としています。本書全体を通して、NGINXの機能とその使用方法についての情報を取得することができます。本ガイドは、包括的な内容となっており、NGINXのほとんどの主要機能について扱います。

本書は、NGINXとNGINX Plusのインストールプロセス、またNGINXを初めて使用する読者のための基本的な開始手順を説明することから始めます。そこから、さまざまな形式のロードバランシング、トラフィック管理、キャッシング、自動化に関する章が続きます。第6章「認証」で多くのスペースを費やしているのは、それが重要であるためです。NGINXは多くの場合、アプリケーションへのWebトラフィックの最初のエントリポイントであり、Web攻撃や脆弱性に対するアプリケーション層の防衛の最前線であるためです。HTTP/3（QUIC）、メディアストリーミング、クラウド、SAML認証、コンテナ環境などの最先端のトピックについてふれる章が多数あります。後半部分では、監視、デバッグ、パフォーマンス、運用上のヒントなど従来の運用関連トピックを扱います。

私は個人的にNGINXをマルチツールとして使用しており、読者の皆様も同じことができると確信しています。これは私が信用し、使用するのを楽しんでいるソフトウェアです。この知識を皆様と共有できることを光栄に感じております。皆様が本書を読みながら、実際のシナリオに合わせ、ソリューションを採用されることを願っています。

本書で使用されている規則

本書では、次の印刷上の規則が使用されています：

斜体
　新しい用語、URL、電子メールアドレス、ファイル名、およびファイル拡張子を示します。

一定幅
　プログラムのリスト、段落内で変数名や関数名、データベース、データ型、環境変数、ステートメント、キーワードなどのプログラム要素に言及するための段落内で使用されます。

この要素は、一般的な注意事項を示します。

この要素は、警告または注意を示します。

O'Reillyオンライン学習

40年以上の間、*O'Reilly Media*が提供してきた技術、ビジネストレーニング、知識、および洞察が企業の成功に役立ちます。

専門家とイノベーターの独自のネットワークは、書籍、記事、オンライン学習プラットフォームを通じて専門知識を共有しています。O'Reillyのオンライン学習プラットフォームを使用すると、ライブトレーニングコース、深く詳細まで扱う課目、インタラクティブなコーディング環境、そしてO'Reillyをはじめ、その他200社以上の出版社からの膨大なテキストとビデオのコレクションにオンデマンドでアクセスできます。詳細については、Webサイトをご覧ください：*http://oreilly.com*.

お問い合わせ方法

この本に関するコメントや質問は、出版社までお送りください。

O'Reilly Media, Inc.
1005 Gravenstein Highway North Sebastopol, CA 95472
800-998-9938（米国またはカナダ）
707-827-7019（国際またはローカル）
707-829-0104 (fax)
support@oreilly.com
https://www.oreilly.com/about/contact.html

本書のWebページがあり、正誤表、例、および追加情報が追加されています。このページには、次のURLからアクセスできます：*https://oreil.ly/nginx-cookbook-3e*

当社の書籍やコースに関するニュース/情報については、次のWebサイトをご覧ください：*https://oreilly.com*

LinkedIn: *https://linkedin.com/company/oreilly-media*

Twitter: *https://twitter.com/oreillymedia*

YouTube: *https://youtube.com/oreillymedia*

第1章
基本

1.0 はじめに

NGINXオープンソースまたはNGINX Plusの使用を開始するには、まずシステムにインストールして、いくつかの基本事項を学ぶ必要があります。本章では、NGINXのインストール方法を学習します。ここで主な構成ファイルの場所、管理用のコマンドについても扱います。また、インストール方法を確認し、デフォルトサーバーにリクエストを送信する方法についても学習します。

本書のレシピの一部ではNGINX Plusを使用しています。NGINX Plusはhttps://nginx.comから無料でお試しいただけます。

1.1 NGINXのDebian/Ubuntuへのインストール

問題

DebianまたはUbuntuマシンにNGINXオープンソースをインストールする必要があります。

解決法

設定されたソースのパッケージ情報を更新し、NGINX公式パッケージリポジトリの設定を行ういくつかのパッケージをインストールします：

```
$ apt update
$ apt install -y curl gnupg2 ca-certificates lsb-release \
    debian-archive-keyring
```

NGINX署名キーをダウンロードし、保存します：

```
$ curl https://nginx.org/keys/nginx_signing.key | gpg --dearmor \
  | tee /usr/share/keyrings/nginx-archive-keyring.gpg >/dev/null
```

`lsb_release`を使用し、OSとリリース名を定義する変数を設定して、*apt*ソースファイルを作成します：

```
$ 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/${OS} ${RELEASE} nginx" \
    | tee /etc/apt/sources.list.d/nginx.list
```

もう一度パッケージ情報を更新して、NGINXをインストールします：

```
$ apt update
$ apt install -y nginx
$ systemctl enable nginx
$ nginx
```

解説

このセクションで使用したコマンドは、Advanced Package Tool（APT）パッケージ管理システムに公式のNGINXパッケージリポジトリを利用するように指示します。NGINX GPGパッケージの署名キーは、APTが使用できるように、ファイルシステム上にダウンロードされ保存されています。APTに署名キーを提供すると、APTシステムはリポジトリからパッケージを検証できるようになります。使用したlsb_releaseコマンドにより、OSとリリース名が自動判別され、DebianやUbuntuのすべてのリリースバージョンでこの指示が使用できるようになります。apt-updateコマンドは、既知のリポジトリからパッケージリストを更新するようにAPTシステムに指示します。パッケージリストが更新されたら、NGINX公式NGINXリポジトリからのオープンソースをインストールできます。インストール後、最後にコマンドでNGINXを起動します。

1.2 YUMパッケージマネージャーを介したNGINXのインストール

問題

NGINXオープンソースをRed Hat Enterprise Linux（RHEL）、Oracle Linux、AlmaLinux、Rocky LinuxまたはCentOSにインストールする必要があります。

解決法

以下のコンテンツを含む*/etc/yum.repos.d/nginx.repo*という名前のファイルを作成します：

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

ファイルを変更し、ディストリビューションに応じてURLの中間に位置するOSを`rhel`または`centos`に置き換えます。その後、以下のコマンドを実行します：

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

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

解説

このソリューション用に作成したファイルは、YUMパッケージ管理システムに公式のNGINXオープンソースパッケージリポジトリを利用するよう指示します。そして以降のコマンドで公式リポジトリからNGINXオープンソースをインストールし、起動時にsystemdでNGINXが起動するよう設定した後、NGINXを起動します。NGINXを有効にし、今すぐ開始するように指示します。必要な場合、ファイアウォールコマンドはHTTPのデフォルトポートであるTransmission Control Protocol（TCP）のポート`80`を開きます。最後のコマンドは、ファイアウォールをリロードして変更をコミットします。

1.3 NGINX Plusのインストール

問題

NGINX Plusをインストールする必要があります。

解決法

NGINX docs (http://cs.nginx.com/repo_setup)にアクセスして、インストールするOSを選択し、指示に従います。手順はオープンソースソリューションのインストールに似ていますが、NGINX Plusリポジトリに対して認証するには、証明書と鍵を取得する必要があります。

解説

NGINXは、NGINX Plusのインストール手順を記載し、このリポジトリインストールガイドを最新の状態に保っています。OSとバージョンによって、これらの手順は若干異なりますが、共通点が1つあります。NGINX Plusリポジトリで認証するには、NGINXポータルから証明書とキーを取得し、システムに提供する必要があります。

1.4 インストールの確認

問題

NGINXのインストールを検証し、バージョンを確認します。

解決法

次のコマンドを使用して、NGINXがインストールされていることとそのバージョンを確認できます。

```
$ nginx -v
nginx version: nginx/1.25.3
```

この例が示すように、出力結果にはバージョンが表示されます。

次のコマンドを使用して、NGINXが実行されていることを確認できます：

```
$ 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
```

psコマンドは、実行中のプロセスを一覧表示します。grepにパイプすることで、特定の用語を出力で検索できます。この例では、grepを使用してnginxを検索します。 結果は、2つの実行中のプロセスを示します：masterとworkerです。NGINXが実行されている場合は、常にマスタープロセスと1つ以上のワーカープロセスが表示されます。マスタープロセスはrootとして実行されていることに注意してください。デフォルトでは、NGINXが適切に機能するためには、一段上の特権が必要です。NGINXの起動手順については、次のレシピを参照してください。NGINXをデーモンとして開始する方法は、init.dまたはsystemdを利用する方法を使用します。

NGINXが要求を正しく返していることを確認するには、ブラウザを使用してリクエストを行うか、curlを使用します。要求を行うときは、対象となるマシンのIPアドレスまたはホスト名を使用してください。ローカルにインストールされている場合は、localhostを以下のように使用できます：

```
$ curl localhost
```

NGINXのデフォルトのWelcome HTMLサイトが表示されます。

解説

nginxコマンドを使用すると、NGINX バイナリを操作して、バージョンを確認したり、インストールされているモジュールを一覧表示したりするほか、構成をテストし、マスタープロセスに信号を送信することができます。要求を処理するには、NGINX が実行されていなければなりません。psコマンドは、NGINXがデーモンとして実行されているかフォアグラウンドで実行されているかを判断する確実な方法です。NGINXでデフォルトで提供される構成は、静的サイトのHTTPサーバーをポート80で実行します。このデフォルトサイトをテストするには、Localhostでマシンに HTTP要求を送信します。ホストのIPとホスト名を使用する必要があります。

1.5 キーファイル、ディレクトリ、コマンド

問題

重要なNGINXディレクトリとコマンドを理解する必要があります。

解決法

以下の構成ディレクトリとファイルの場所は、NGINXのコンパイル中に変更できるので、インストールによって異なる場合があります。

NGINXファイルとディレクトリ

/etc/nginx/

/etc/nginx/ ディレクトリは、NGINXサーバーのデフォルトの構成ルートです。このディレクトリ内には、NGINXに動作方法を指示する構成ファイルがあります。

/etc/nginx/nginx.conf

*/etc/nginx/nginx.conf*ファイルは、NGINXデーモンサービスによって使用されるデフォルトの構成エントリポイントです。この構成ファイルは、次のような機能のグローバル設定をセットアップします：ワーカープロセス、チューニング、ロギング、動的モジュールの読み込み、他のNGINX構成ファイルへの参照。デフォルトの構成では、*/etc/nginx/nginx.conf* ファイルにはトップレベル`http`ブロック、またはコンテキストが含まれます。これには、次に説明するディレクトリ内のすべての構成ファイルが含まれます。

/etc/nginx/conf.d/

/etc/nginx/conf.d/ ディレクトリにはデフォルトのHTTPサーバーの構成ファイルが含まれます。このディレクトリ内のファイルで*.conf*で終わるものは*/etc/nginx/nginx.conf*ファイル内の`http`トップレベルに含まれています。これは、`include`ステートメントを活用し、構成を整理し、構成ファイルを簡潔に保つためのベストプラクティスです。一部のパッケージリポジトリでは、このフォルダの名前は *sites-enabled* という名前で、*site-available* という名前のフォルダからリンクされています。この規則は非推奨です。

/var/log/nginx/

/var/log/nginx/ ディレクトリは、NGINXのデフォルトのログの場所です。このディレクトリ内に*access.log*ファイルと*error.log*ファイルがあります。デフォルトでは、アクセスログには、NGINXが提供する各要求のエントリが含まれています。エラーログファイルには、エラーイベントとdebugモジュールが有効になっている場合にはデバッグ情報が含まれています。

NGINXコマンド

`nginx -h`

NGINXヘルプメニューを表示します。

`nginx -v`

NGINXバージョンを表示します。

`nginx -V`

NGINXバージョン、ビルド情報、NGINXバイナリに組み込まれているモジュールを示す構成引数を表示します。

`nginx -t`

NGINX構成をテストします。

`nginx -T`

NGINX構成をテストし、検証された構成を画面に出力します。このコマンドは、サポートを求めるときに役立ちます。

`nginx -s signal`
: `-s`フラグはNGINXマスタープロセスに信号を送ります。`stop`、`quit`、`reload`、`reopen`のような信号を送信できます。`stop`信号はNGINX プロセスを即座に中止します。 `quit`信号はNGINXプロセスを処理中のリクエストの処理後に停止します。 `reload`信号は構成をリロードします。`reopen`信号は、ログファイルを再度開くようにNGINXに指示します。

解説

これらの主要なファイル、ディレクトリ、コマンドを理解することで、NGINXの使用を順調に開始できます。この知識があれば、デフォルトの構成ファイルを変更して、その後`nginx -t`コマンドを使用してテストすることができます。テストが成功した場合は、`nginx -s reload`コマンドを使用して、NGINXに構成を再読み込みするよう指示する方法も説明されました。

1.6 includeを使用した構成の整理

問題

大容量の構成ファイルを整理して、構成をモジュラー構成セットに論理的にグループ化する必要があります。

解決法

構成ファイル、ディレクトリまたはマスクを参照する`include`ディレクティブを使用します：

```
http {
  include conf.d/compression.conf;
  include ssl_config/*.conf
}
```

`include`ディレクティブは、1つのファイルへのパス、または複数のファイルにマッチするマスクのどちらかをパラメータとして使用します。このディレクティブは任意のコンテキストで有効です。

解説

`include`ステートメントを使用して、NGINX構成を簡潔に整理できます。構成を論理的にグループ化し、数百行もある構成ファイルを回避できます。構成全体の複数の場所に含めることができるモジュラー構成ファイルを作成して、構成の重複を避けることができます。

NGINXのほとんどのパッケージ管理インストールで提供される*fastcgi_param*構成ファイルを例に挙げて説明します。1つのNGINXボックスで複数のFastCGI仮想サーバーを管理する場合、この構成が重複することなく、FastCGIのパラメータが必要な場所やコンテキストにこの構成ファイルを含めることができます。もう1つの例として、Secure Sockets Layer（SSL）構成を例に挙げます。同じようなSSL構成を必要とする複数のサーバーを運用している場合、この構

成を一度作成するだけで、必要なところに含めることができます。

構成を論理的にグループ化することで、構成を簡潔に整理できます。大容量の構成ファイル内の複数の場所にある構成ブロックのセットを複数変更するのではなく、単一のファイルを編集することで、構成ファイルの1つのセットを変更できます。ご自身と同僚の健全性のために、構成をファイルにグループ化し、`include`ステートメントを使用することをお勧めします。

1.7 静的コンテンツの提供

問題

NGINXで静的コンテンツを提供する必要があります。

解決法

*/etc/nginx/conf.d/ default.conf*にあるデフォルトのHTTPサーバー構成を以下のNGINX構成例に置換します：

```
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;
  }
}
```

解説

この構成は、ディレクトリ*/usr/share/nginx/html/*からポート`80`でHTTP経由で静的ファイルを提供します。 この構成の最初の行で、新しいサーバーブロックを定義します。これはNGINXがリッスンする新しいコンテキストを定義します。2行目はNGINXにポート`80`をリッスンするよう指示し、`default_server`パラメータはNGINXにこのサーバーをポート`80`のデフォルトのコンテキストとして使用するよう指示します。`listen`ディレクティブはポートの範囲を指定することもできます。`server_name`ディレクティブは、このサーバーに要求を送信するホスト名を定義します。

構成でこのコンテキストを`default_server`として定義しないと、NGINXは要求をHTTPホストヘッダーが`server_name`ディレクティブで提供された値と一致する場合にのみ要求をこのサーバーにダイレクトします。`default_server`コンテキストセットがあれば、使用するドメイン名がない場合には`server_name`ディレクティブを省略できます。

`location`ブロックはURLに含まれるパスに基づいて、構成を定義します。ドメインに続くURL

の一部であるこのパスはuniform resource identifier（URI）と呼ばれます。NGINXは`location`ブロックに要求されたものにURIを一致させるようにします。この例はすべてのリクエストを一致させるために/を使用しています。`root`ディレクティブは、指定されたコンテキストのコンテンツを提供するときに静的ファイルを探す場所をNGINXに示します。要求されたファイルを探す際に、リクエストのURIを`root`ディレクティブの値に追加されます。また、`location`ディレクティブに入力したURIプレフィックスは、同様に対象となるパスに追加されます。ただし、`root`ディレクティブではなく`alias`ディレクティブを使用した場合にはこれらのパスの追加は行われません。`location`ディレクティブは、さまざまな表現に一致することができます。詳細については、「関連項目」セクションの最初のリンクを参照してください。最後に、`index`ディレクティブは、URIにそれ以上のパスが指定されていない場合に、NGINXにデフォルトファイルまたはチェックするファイルのリストを提供します。

関連項目

NGINX HTTP location ディレクティブドキュメンテーション

NGINXリクエスト処理

第2章
ハイパフォーマンスロードバランシング

2.0 はじめに

今日のインターネットユーザーエクスペリエンスには、パフォーマンスとアップタイムが必要です。この目的を達成するため、同一システムで複数のコピーを実行し、そしてそれらのコピー上で負荷を分散することができます。負荷が増加したときに、更に別のコピーをオンラインとして実行することができます。このアーキテクチャ手法は、*水平スケーリング*と呼ばれます。ソフトウェアベースのインフラストラクチャは、その柔軟性のために人気が高まっており、幅広い可能性の世界が広がっています。高可用性のための2つのシステムコピーのセットという小規模なものから、世界中にある何千ものシステムコピーという大規模なものまで、インフラと同じようにダイナミックな負荷分散ソリューションが必要とされているのです。NGINXは、HTTP、Transmission Control Protocol（TCP）、ユーザーデータグラムプロトコル（UDP）の負荷分散など、本章で説明されるさまざまな方法でこのニーズを満たします。

ロードバランシングに関しては、クライアントの体験に完全にポジティブな影響を与えることが重要です。最新のWebアーキテクチャの多くは、ステートレスアプリケーションでの実装を採用しており、状態を共有メモリまたはデータベースに格納しています。しかし、これはすべての人にとって現実的な実装であるというわけではありません。セッション状態は、インタラクティブなアプリケーションでは非常に価値があり広く使用されています。この状態は、いくつかの理由でアプリケーションサーバーのローカルに保存される可能性があります。たとえば、アプリケーションのために処理中のデータが非常に大きいため、ネットワークのオーバーヘッドのパフォーマンスが高すぎる場合などです。状態がアプリケーションサーバーにローカルに保存される場合、後続のリクエストが同じサーバーに配信され続けることがユーザーエクスペリエンスにとって非常に重要です。この状況のもう1つの側面は、セッションが終了するまでサーバーを解放しないことです。大規模なステートフルアプリケーションを操作するには、インテリジェントなロードバランサーが必要です。NGINXPlusは、Cookieを追跡したりルーティングしたりすることで、この問題を解決する複数の方法を提供します。本章では、NGINXを使用したロードバランシングに関連するセッションの永続性について説明します。

NGINX がサービスを提供しているアプリケーションが正常であることを確認することが重要です。多くの理由から、アップストリームリクエストが失敗し始める場合があります。これは、ネットワーク接続、サーバーの故障、アプリケーションの故障などが原因かもしれません。プロキシとロードバランサーはアップストリームサーバー（ロードバランサーまたはプロキシの背後にあるサーバー）の障害を検出し、故障したサーバーへのトラフィックの受け渡しを停止できるスマートさが必要です。これができなければ、クライアントは待機し続け、タイムアウトされるだけです。

サーバーに障害が発生した場合のサービスの低下を軽減する方法は、プロキシにアップストリームサーバーの状態をチェックさせることです。 NGINX は2 つの異なるタイプのヘルスチェックを提供します。NGINX オープンソースで利用可能なパッシブヘルスチェックとNGINX Plus でのみ利用可能なアクティブヘルスチェックです。定期的なアクティブヘルスチェックは、アップストリームサーバーへの接続または要求を行い、応答が正しいことを確認します。パッシブヘルスチェックは、クライアントが要求または接続を行うときに、アップストリームサーバーの接続または応答を監視します。パッシブヘルスチェックを使用してアップストリームサーバーの負荷を軽減し、アクティブヘルスチェックを使用して、クライアントに障害が発生する前にアップストリームサーバーの障害を特定することができます。本章の最後では、ロードバランシングを行っているアップストリームアプリケーションサーバーの状態の監視について説明します。

2.1 HTTP負荷分散

問題

2つ以上のHTTPサーバー間で負荷を分散する必要があります。

解決法

HTTPサーバー間のロードバランシングにNGINXのHTTPモジュールのupstreamブロックを使用します。

```
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;
  }
}
```

この構成では、80番ポートの2台のHTTPサーバーに負荷を分散し、1台をバックアップとして定義し、バックアップはプライマリサーバー 2台が使用できないときに使用します。オプションのweightパラメータはNGINXに対して2つ目のサーバーに2倍のリクエストを渡すよう指示

します。使用されない場合、weightパラメータのデフォルトは1です。

解説

HTTP upstreamモジュールはHTTPリクエストのロードバランシングを制御します。このモジュールは宛先のプールを定義します。Unixソケット、IPアドレス、サーバーホスト名、または任意の組み合わせが宛先となります。 upstreamモジュールは、個々の要求をアップストリームサーバーに割り当てる方法も定義します。

各アップストリーム宛先は、アップストリームプールでserverディレクティブによって定義されます。serverディレクティブは、アップストリームサーバーのアドレスとともに、オプションのパラメータも受け取ります。オプションのパラメータを使用すると、要求のルーティングをより細かく制御できます。これらのパラメータには、バランシングアルゴリズムにおけるサーバーのウェイトが含まれます。サーバーがスタンバイモードであるか、使用可能であるか、使用不可であるか、そして、サーバーが利用できないかどうかを判断する方法です。NGINX Plusは、サーバーへの接続制限、高度なDNS解決制御、サーバーの起動後にサーバーへの接続をゆっくりと立ち上げる機能など、他の多くの便利なパラメータを提供します。

2.2 TCPロードバランシング

問題

2つ以上のTCPサーバー間で負荷を分散する必要があります。

解決法

NGINXのstreamモジュールのupstreamブロックを使用して、TCPサーバー間のロードバランスを行います。

```
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;
  }
}
```

この例のserverブロックは、NGINXにTCPポート3306でリッスンし、2つのMySQLデータベース読み取りレプリカ間の負荷を分散し、プライマリがダウンした場合にトラフィックが渡されるバックアップとして別のレプリカをリストします。

この構成は*conf.d*フォルダに追加されません。このフォルダはデフォルトでhttpブロックに含まれています。そのため、*stream.conf.d*という名前の別のフォルダを作成し、*nginx.conf*ファイルでstreamブロックを開き、ストリーム構成のための新しいフォルダを含める必要があります。以下に例を示します。

/etc/nginx/nginx.conf 構成ファイル内に：

```
user nginx;
worker_processes auto;
pid /run/nginx.pid;

stream {
  include /etc/nginx/stream.conf.d/*.conf;
}
```

*/etc/nginx/stream.conf.d/mysql_reads.conf*という名前のファイルの中に、構成ファイル：

```
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;
}
```

解説

httpおよびstreamコンテキストの主な違いはOSIモデルの異なる層で動作する点です。httpコンテキストは第7層であるアプリケーション層で動作し、およびstreamは第4層であるトランスポート層で動作します。これは、streamコンテキストは巧妙なスクリプトを用いてアプリケーション対応できないということを意味するわけではありません。httpコンテキストはHTTPプロトコルを完全に理解するよう特別に設計されていて、streamコンテキストはデフォルトでパケットを単にルーティングして、負荷分散するものだということです。

TCPロードバランシングはNGINX streamモジュールによって定義されます。streamモジュールはHTTPモジュールと同様にサーバーのアップストリームプールを定義し、リッスンするサーバーを構成します。特定のポートでリッスンするようにサーバーを構成するときは、リッスンするポート、またはオプションでアドレスとポートを定義する必要があります。そこから、別のアドレスへの直接リバースプロキシか、上流のリソースのプールへ宛先を構成する必要があります。

TCP接続のリバースプロキシのプロパティを変更するために多数のオプションを構成で利用することができます。その一部には、SSL/TLS検証の制限、タイムアウト、およびキープアライブが含まれます。これらのプロキシオプションの値の一部は、ダウンロードレートや、SSL/TLS証明書の検証に使用される名前などの変数にすることができます（または変数を含むこと

ができます)。

TCPロードバランシングのアップストリームは、HTTPのアップストリームとよく似ています。アップストリームリソースをサーバーとして定義し、Unixソケット、IP、FQDN、さらにサーバーウェイト、接続の最大数、DNSリゾルバ、接続ランプアップ時間で構成されるほか、サーバーがアクティブか、ダウンか、バックアップモードかで構成できます。

NGINX Plusは、TCPロードバランシングのためのさらに多くの機能を提供します。これらの高度な機能は、本書全体で紹介されます。すべてのロードバランシングのヘルスチェックについては、本章の後半で説明します。

2.3 UDPロードバランシング

問題

2つ以上のUDPサーバー間で負荷を分散する必要があります。

解決法

NGINXの`stream`モジュールの`udp`として定義された`upstream`ブロックを使用してUDPサーバー上でロードバランスします:

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

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

構成のこのセクションでは、UDPプロトコルを使用した2つのアップストリームネットワーク時間プロトコル(NTP)サーバー間でのロードバランスを実行します。UDPロードバランシングの指定は、`udp`のパラメータを`listen`ディレクティブで使用する場合と同じくらいシンプルです。

ロードバランシングを行うサービスで、クライアントとサーバー間で複数のパケットを送受信する必要がある場合は、`reuseport`パラメータを指定できます。これらのタイプのサービスの例としては、OpenVPN、Voice over Internet Protocol (VoIP)、virtual desktop solutions、Datagram Transport Layer Security (DTLS) 等が挙げられます。以下はNGINXを使用してOpenVPN接続を処理し、ローカルで実行されているOpenVPNサービスにプロキシする場合の例です:

```
stream {
  server {
    listen 1195 udp reuseport;
```

```
        proxy_pass 127.0.0.1:1194;
    }
}
```

解説

あなたは「DNS AまたはサービスレコードSRVレコードに複数のホストをもてるのに、なぜロードバランサーが必要なのですか？」と質問するかもしれません。その質問に対する答えは、負荷分散できる別のバランシングアルゴリズムがあるからというだけではなく、DNSサーバー自体のロードバランシングもできるから、ということになります。UDPサービスは、DNS、NTP、QUIC、HTTP / 3、VoIPなど、ネットワークシステムに依存する多くのサービスを構成しています。UDPロードバランシングは、一部の人にはあまり一般的ではないかもしれませんが、大規模環境でも同様に役立つことがあります。

UDPロードバランシングはTCP同様にstreamモジュールで見つけることができ、ほとんど同じ方法で構成します。主な違いは、listenディレクティブが、オープンソケットがデータグラムを操作するためのものであることを指定する点です。データグラムを操作する場合、TCPでは特に指定しない場所で適用できるディレクティブがいくつかあります。アップストリームサーバからNGINXへ送付されるレスポンス数の予測した値を指定するproxy_responce等があります。デフォルトでは、これはproxy_timeout制限に達するまで無制限です。proxy_timeoutディレクティブはコネクションがクローズする前にプロキシサーバやクライアントが操作する連続する２つの読み込みまたは書き込みの間の時間を指定します。

reuseportパラメータは、ワーカープロセスごとに個別のリスニングソケットを作成するようNGINXに指示します。この機能により、カーネルはクライアントとサーバ間に置いてやり取りされるパケットをワーカープロセス間の受信するコネクションに分散する事ができます。reuseport機能は、Linuxカーネル3.9以降、DragonFly BSD、および FreeBSD12以降でのみ機能します。

2.4 ロードバランシング方法

問題

異なる種類ののワークロードまたはサーバープールがあるため、ラウンドロビンロードバランシングがユースケースに適合しません。

解決法

最小接続、最短時間、ジェネリックハッシュ、ランダム、またはIPハッシュなどNGINXのロードバランシング方法の１つを使用します。

この例では、バックエンドアップストリームプールの負荷分散アルゴリズムを最小接続数のサーバーを選択するように設定します：

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

ジェネリックハッシュ、ランダム、および最短時間を除くすべての負荷分散アルゴリズムは、前述の例のようにスタンドアロンのディレクティブです。これらのディレクティブのパラメータについては、以下の解説で説明します。

次の例では、`$remote_addr`変数でジェネリックハッシュアルゴリズムを使用します。この例は、IPハッシュと同じルーティングアルゴリズムをレンダリングします。ただし、ジェネリックハッシュは`stream`コンテキストで動作しますが、IPハッシュは`http`コンテキストでのみ使用可能です。使用する変数を置き換えたり、追加したりして、ジェネリックハッシュアルゴリズムの負荷分散方法を変更できます。以下は、ジェネリックハッシュアルゴリズムでクライアントのIPアドレスを使用するように設定された`upstream`ブロックの例です：

```
upstream backend {
  hash $remote_addr;
  server backend.example.com;
  server backend1.example.com;
}
```

解説

すべての要求またはパケットに同じ重みがあるわけではありません。これを考慮すると、ラウンドロビン、または前の例で使用した重み付きラウンドロビンでさえ、すべてのアプリケーションまたはトラフィックフローのニーズに適合するわけではありません。 NGINXは、特定のユースケースに適合させるために使用できる多数の負荷分散アルゴリズムを提供します。これらを選択できることに加えてロードバランシングアルゴリズムまたはメソッドを選択し、構成することもできます。次のロードバランシング方法は、IPハッシュの例外はありますが、アップストリームHTTP、TCP、およびUDPプールで使用できます：

ラウンドロビン

これはデフォルトのロードバランシング方法であり、アップストリームプール内のサーバーのリストの順番で要求を分散します。重み付きラウンドロビンの場合、重みを考慮し、アップストリームサーバーの容量がそれぞれ異なる場合に使用することができます。重みの整数値が高いほど、そのサーバーはラウンドロビンにとって好まれることになります。重みの背後にあるアルゴリズムは 単に統計による重みの平均の確率です。

最小接続数

この方法では、接続中コネクションの数が最も少ないアップストリームサーバーに現在の要求をプロキシすることにより、負荷を分散します。最小接続数での分散は、どのサーバにコネクションを転送するか決定する際、ラウンドロビンのように重みを考慮します。ディレクティブ名は`least_conn`です。

最短時間

NGINX Plusでのみ利用可能な最短時間での分散は現在の接続数が最も少ないアップストリームサーバーにプロキシする最小接続数に似ていますが、最短時間では、平均応答時間が最短となるサーバーが選択されます。この方法は、最も洗練された負荷分散アルゴリズムの1つであり、高パフォーマンスWebアプリケーションのニーズに適合します。このアルゴリズムは最小接続数に対する付加価値として使用されます。接続数が最小であるからといって、応答時間が最速であるとは限らないためです。このアルゴリズムを使用する際には、サービスの要求時間の統計的差異を考慮することが重要です。一部の要求はより多くの処理を必要とするため、要求時間が長くなり、統計上値が増加します。要求時間が長いからといって、サーバーのパフォーマンスが低かったり、サーバーが過負荷になっていたりするわけではありません。しかし、より多くの処理が必要な要求は非同期ワークフローが適しているかもしれません。このディレクティブではheaderまたは`last_byte`パラメータを指定しなければならなりません。`header`が指定された場合、応答ヘッダーを受信するまでの時間が使用されます。`last_byte`が指定された場合、完全な応答を受信するまでの時間が使用されます。`inflight`パラメータが指定されている場合、不完全なリクエストも考慮されます。ディレクティブ名は`least_time`です。

ジェネリックハッシュ

管理者は、指定されたテキストと要求またはランタイムのいずれかの変数、あるいはその両方を使用してハッシュを定義します。NGINXは、現在の要求のハッシュを生成し、それをアップストリームサーバーに対して配置することで、サーバー間で負荷を分散します。この方法は、要求の送信先をより細かく制御する必要がある場合、またはデータがキャッシュされる可能性が最も高いアップストリームサーバーを決定する場合に非常に役立ちます。サーバーがプールに追加またはプールから削除されると、ハッシュされた要求が再構成されることに注意してください。このアルゴリズムには再構成の影響を最小限に抑える役割をするオプションのパラメータ`consistent`があります。ディレクティブ名は`hash`です。

ランダム

この方法は、NGINXにサーバーの重みを考慮してグループからランダムにサーバーを選択するように指示するために使用されます。オプションの2つの *[method]* パラメータはNGINXに2つのサーバーをランダムに選択して、提供されたロードバランシング方法を使用して、この2つの間で負荷を分散するよう指示します。`two`がメソッドなしで渡され場合、デフォルトでは`least_conn`方法が使用されます。ランダムロードバランシングのディレクティブ名は`random`です。

IPハッシュ

この方法はHTTPでのみ機能します。IPハッシュは、クライアントのIPアドレスをハッシュとして使用します。このアルゴリズムは、ジェネリックハッシュでリモート変数を使用する場合とは少し異なり、IPv4アドレスの最初の3オクテットまたはIPv6アドレス全体を使用します。この方法はサーバーが使用可能である限り、クライアントが同じアップストリームサーバーにプロキシされるようにします。これは、セッション状態が懸念され、アプリケーションの共有メモリによって処理されない場合に非常に役立ちます。この方法でも、ハッシュを

分散する際にweightパラメータが考慮されます。ディレクティブ名はip_hashです。

2.5 NGINX Plusを使用したスティッキークッキー

問題

NGINX Plusを使用して、ダウンストリームクライアントをアップストリームサーバーにバインドする必要があります。

解決法

sticky cookieディレクティブを使用してNGINX Plusにcookieを作成、追跡するよう指示します。

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

この構成は、ダウンストリームクライアントをupstreamサーバーにバインドするcookieを作成および追跡します。この例では、cookieはaffinityと名づけられ、*example.com*に設定されており、1時間で期限切れになり、クライアント側で使用できず、HTTPS経由でのみ送信でき、すべてのパスで有効です。

解説

cookieパラメータをstickyディレクティブで使用すると、アップストリームサーバーに関する情報が含まれる最初の要求でcookieが作成されます。NGINX PlusはこのCookieを追跡し、後続の要求を同じサーバーに送信し続けます。cookieパラメータの最初の位置にあるパラメータは作成および追跡されるcookieの名前です。他のパラメータは追加の制御を提供します。ブラウザに適切な使用-有効期限、ドメイン、パスを知らせたり、cookieをクライアント側で消費できるか、またはそれを安全でないプロトコルで渡すことができるかどうかなどの情報を提供します。

CookiesはHTTPプロトコルの一部であるため、sticky cookieはhttpコンテキストのみで機能します。

2.6 NGINX Plusを使用したスティッキーラーン

問題

NGINX Plusで既存のcookieを使用してダウンストリームクライアントをアップストリームサーバーにバインドする必要があります。

解決法

`sticky learn`ディレクティブを使用して、アップストリームアプリケーションにより作成されたcookiesを発見・追跡します。

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

  sticky learn
         create=$upstream_cookie_cookiename
         lookup=$cookie_cookiename
         zone=client_sessions:1m;
}
```

この例では、NGINXに、応答ヘッダーで`COOKIENAME`という名前のcookieを探してセッションを追跡し、同じcookieを要求ヘッダーで探して既存のセッションをルックアップするよう指示します。このセッションアフィニティは約4,000セッションを追跡できる容量1 MBの共有メモリーゾーンに保存されます。Cookieの名前は、常にアプリケーション固有です。一般的に使用される`jsessionid`または`phpsessionid`のようなCookieは通常、アプリケーションまたはアプリケーションサーバー構成内で設定されるデフォルトです。

解説

アプリケーションが独自のセッションステートcookieを作成すると、NGINX Plusは要求への応答でそれらを検出し、追跡できます。このタイプのcookie追跡は、`sticky`ディレクティブが`learn`パラメータとともに提供されたときに実行されます。cookieを追跡するための共有メモリは、`zone`パラメータ、名前、容量とともに指定されます。NGINX Plusは、`create`パラメータの指定を介してアップストリームサーバーからの応答で`cookie`を探し、`lookup`パラメータを使用して以前登録されたサーバーアフィニティを検索するよう指示されます。これらのパラメータの値は、`HTTP`モジュールにより指定された変数です。

2.7 NGINX Plus を使用したスティッキールーティング

問題

NGINX Plusを使用して、永続セッションをアップストリームサーバーにルーティングする方法をきめ細かく制御する必要があります。

解決法

要求のルーティングに関する変数を使用するためのrouteパラメータのあるstickyディレクティブの使用：

```
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;
}
```

この例では、最初にJavaセッションID cookieの値を最初のmapブロックを持つ変数にマッピングすることにより、最初にcookieからJavaセッションIDを抽出しようとします。次に2番目のmapブロックを使用して値を変数にマッピングして、jsessionidというパラメータを要求URIを検索して見つけようとします。routeパラメータを伴うstickyディレクティブには任意の数の変数が渡されます。最初のゼロではない、または空でない値がルートに使用されます。もし jsessionid cookieに特定の値が使用された場合には、要求はbackend1にルーティングされる場合があり、URIパラメータに特定の値が使用されている場合、要求はbackend2にルーティングされます。この例はJava共通セッションIDに基づいていますが、同じことがphpsessionid、または、アプリケーションがセッションIDに生成する保証された一意の識別子のような他のセッションテクノロジーにも当てはまります。

解説

場合によっては、もう少しきめ細かい制御を利用して、トラフィックを特定のサーバーに転送したいことがあります。stickyディレクティブへのrouteパラメータは、この目標を達成するために作成されています。スティッキールートは、ジェネリックハッシュロードバランシングアルゴリズムとは異なり、より優れた制御、実際の追跡、Stickinessを提供します。クライアントは指定されたルートに基づき、まず、アップストリームサーバーにルーティングされ、次に、後続の要求はルーティング情報をCookieまたはURIに含みます。スティッキールートは評価に利用できる位置を示すパラメータをとります。空でない最初の変数は、サーバーへのルーティ

ングに使用されます。マップブロックを使用して、変数を選択的に解析し、ルーティングで使用するその他の変数として保存できます。`sticky route`ディレクティブはNGINX Plusの共有メモリーゾーン内にセッションを作成します。これはアップストリームサーバーのために指定されたクライアントセッションの識別子を追跡するためであり、また一貫してセッションの識別子を持ったリクエストを同じアップストリームサーバーに元のリクエストとして転送するためです。

2.8 NGINX Plusを使用した接続のドレイン

問題

NGINX Plusとのセッションを提供しながら、メンテナンスまたはその他の理由でサーバーを適切に削除する必要があります。

解決法

NGINX Plus APIを介して`drain`パラメータを使用してNGINXにまだ追跡されていない新しい接続を送信するのを停止するよう指示します。（詳細な説明は第5章を参照。）

```
$ curl -X POST -d '{"drain":true}' \
  'http://nginx.local/api/9/http/upstreams/backend/servers/0'

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

解説

セッション状態がサーバーにローカルに保存されている場合、サーバーをプールから削除する前に、接続と永続セッションをドレインする必要があります。ドレイン接続はサーバーをアップストリームプールから削除する前にサーバーへのセッションをネイティブに期限切れにする方法のプロセスです。特定のサーバーのドレインは`server`ディレクティブに`drain`パラメータを追加することで構成できます。`drain`パラメータが設定されるとNGINXPlusは新しいセッションをこのサーバーに送るのを停止しますが現在のセッションがセッション中は継続を許可します。この構成は`drain`パラメータをアップストリームサーバーのディレクティブに追加し、

NGINX Plus構成をリロードすることで変更できます。

2.9 パッシブヘルスチェック

問題

プロキシされたトラフィックを正常に処理しているかどうか、アップストリームサーバーの正常性を受動的（パッシブ）にチェックする必要があります。

解決法

NIGNXヘルスチェックをロードバランシングと一緒に使用して、健全なアップストリームサーバーのみが利用されるようにします：

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

この構成は、アップストリームサーバーに向けられたクライアント要求の応答を監視することにより、アップストリームの状態をパッシブに監視します。この例では、`max_fails`ディレクティブを3に、そして`fail_timeout`を3秒に設定します。これらのディレクティブパラメータは、ストリームサーバーとHTTPサーバーの両方で同じように機能します。

解説

パッシブヘルスチェックは、NGINXオープンソースで利用できます。HTTP、TCP、およびUDPロードバランシングのサーバーパラメータを使用して構成されます。パッシブモニタリングは、クライアントの要求に応じてNGINXを通過する時に、接続の失敗またはタイムアウトを監視します。パッシブヘルスチェックは、デフォルトで有効で、ここに記載されているパラメータを使用すると、動作を微調整できます。デフォルトの`max_fails`値は1、デフォルトの`fail_timeout`値は`10s`です。ヘルスモニタリングは、ユーザーエクスペリエンスの観点からだけでなく、ビジネス継続性の観点からも、すべてのタイプのロードバランシングで重要です。NGINX は、アップストリームのHTTP、TCP、およびUDP サーバーをパッシブに監視して、それらが正常で実行されていることを確認します。

関連項目

HTTPヘルスチェック管理者ガイド

TCPヘルスチェック管理者ガイド

UDPヘルスチェック管理者ガイド

2.10 NGINX Plusのアクティブヘルスチェック

問題

プロキシされたトラフィックを正常に処理しているかどうか、NGINX Plusを使用して、アップストリームサーバーの正常性を積極的（アクティブ）にチェックする必要があります。

解決法

HTTPの場合は、health_checkディレクティブを以下のロケーションブロックで使用します：

```
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!";
  }
}
```

このHTTPサーバーのヘルスチェック構成は、2秒ごとにURI "/" に対してHTTPGET要求を行うことにより、アップストリームサーバーのヘルスをチェックします。他の方法ではバックエンドシステムの状態を変更する可能性があるため、ヘルスチェックではHTTPメソッドを定義できず、GETリクエストのみを実行します。アップストリームサーバーが正常と見なされるためには、5回連続してヘルスチェックに合格する必要があります。アップストリームサーバーは、2回連続でチェックに失敗すると不健全とみなされ、プールから外されます。アップストリームサーバーからの応答は、次のように定義する定義済みの一致ブロックと一致する必要があります：ステータスコード200、ヘッダーContent-Typeを'text/ html'、応答本文で文字列"Welcome to nginx!"。HTTP matchブロックには3つのディレクティブがあります。これら3つのディレクティブにはすべて、比較フラグも利用しています。

TCP/UDPサービスのストリームヘルスチェックは非常に似ています：

```
stream {
  # ...
  server {
```

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

この例では、TCPサーバーはポート1234をリッスンし、アクティブヘルスチェックを実行するupstreamサーバーセットにプロキシするよう構成されます。ストリームhealth_checkディレクティブはHTTPと同じパラメータを使用します（例外はuri）が、ストリームバージョンにはチェックプロトコルをudpに変更するためのパラメータがあります。この例では、間隔は10秒に設定されていて、健全であると認められるためには2つの連続するチェックをパスする必要があり、3回失敗すると不健全であるとみなされます。アクティブストリームヘルスチェックは、アップストリームサーバーからの応答も検証できます。しかし、ストリームサーバーのmatchブロックにはsendとexpectの2つのディレクティブしかありません。sendディレクティブは生データを送信し、expectは完全一致の応答または一致する正規表現です。

解説

NGINX Plusでは、パッシブまたはアクティブヘルスチェックはソースサーバーを監視するために使用できます。これらのヘルスチェックは応答コード以上のものを計測できます。NGINXPlusでは、アクティブHTTPヘルスチェックモニターはアップストリームサーバーへの応答受け入れ条件の正常応答回数に基づいています。アップストリームサーバーがチェックされる頻度、サーバーが正常であると認められるためには何度チェックをパスする必要があるか、何回までなら異常と認められずに失敗できるか、期待される結果はなにかなどをモニターするためにアクティブヘルスチェックを構成できます。より複雑なロジックでは、matchブロックへのrequireディレクティブが変数の使用を可能にします。変数の値はemptyまたはゼロには設定できません。matchパラメータは応答の受け入れ基準を定義するmatchブロックを指します。matchブロックは、TCP/UDPのストリームコンテキストで使用されるときにアップストリームサーバーに送信するデータも定義します。これらの機能はNGINXがアップストリームサーバーの健全性を常に保証できるようにします。

関連項目

HTTPヘルスチェック管理者ガイド

TCPヘルスチェック管理者ガイド

UDPヘルスチェック管理者ガイド

2.11 NGINX Plusのスロースタート

問題

使用しているアプリケーションは本番トラフィックのすべてを受け付ける前に段階的に動作を開始する必要があります。

解決法

serverディレクティブにslow_startパラメータを使用して、指定された時間内に接続の数を徐々に増やしながら、サーバーをアップストリームロードバランシングプールに再度導入していきます：

```
upstream {
  zone backend 64k;

  server server1.example.com slow_start=20s;
  server server2.example.com slow_start=15s;
}
```

このserverディレクティブの構成はアップストリームサーバーを再導入後、トラフィックをゆっくりと増加させていきます。server1は20秒、server2は15秒かけて接続の数を増やします。

解説

*スロースタート*は、サーバーにプロキシされる要求の数を一定期間にわたってゆっくりと増加させるという考え方です。スロースタートを使用すると、アプリケーションは起動直後に接続に圧倒されることなく、キャッシュにデータを入力してウォームアップし、データベース接続を開始できます。この機能は、ヘルスチェックにいったん失敗したサーバーが再びヘルスチェックに合格してロードバランシングプールに再び加わる際に有効になります。また、この機能はNGINX Plusでのみ使用できます。スロースタートは hash、ip_hash、randomロードバランシング方法では使用できません。

第3章
トラフィック管理

3.0 はじめに

NGINXは、Webトラフィックコントローラとしても分類できます。NGINXを使用して、トラフィックをインテリジェントにルーティングし、多くの属性に基づいてフローを制御できます。本章では、指定した割合に基づいてクライアントリクエストを分割するNGINXの機能について説明します。クライアントの地理的位置を利用する機能のほか、レート、接続、帯域幅の制限という形でトラフィックの流れを制御します。本章を読み進めるうちに、これらの機能を組み合わせて無数の可能性を実現できることにお気づきになるはずです。

3.1 A/Bテスト

問題

受け入れまたはエンゲージメントをテストするために、ファイルまたはアプリケーションの2つ以上のバージョン間でクライアントを分割する必要があります。

解決法

`split_clients`モジュールを使用して、複数のクライアントをパーセンテージごとに異なるアップストリームプールにダイレクトします。

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

`split_clients`ディレクティブは最初のパラメータとして提供された文字列をハッシュし、そのハッシュを提供されたパーセンテージで除算して、2番目のパラメータとして提供された変数の値をマップします。最初のパラメータに「AAA」を追加することは、ジェネリックハッシュロードバランシングアルゴリズムで説明されているように、これが多くの変数を含むことができる連結文字列であることを示すためです。3番目のパラメータは、キーと値のペアを含むオブジェ

クトです。ここで、キーは重みのパーセンテージであり、値はキーによって割り当てられる値です。キーはパーセンテージまたはアスタリスクを使用できます。アスタリスクは、すべてのパーセンテージが取られた後の残りを示します。$variant変数の値はクライアントIPアドレスの20%についてはbackendv2、残りの80%はbackendv1になります。

この例では、backendv1およびbackendv2はアップストリームサーバープールを表し、proxy_passディレクティブと併用できます：

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

変数$variantを使用することで、トラフィックは2 つの異なるアプリケーションサーバープールに分割されます。

split_clientsが可能にするさまざまな使用例を示すために、以下に2つのバージョンの静的サイト間で分割する場合の例を示します：

```
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;
    }
  }
}
```

解説

この種類のA/Bテストは、eコマースサイトでのコンバージョン率について、さまざまなタイプのマーケティングやフロントエンド機能をテストする場合に役立ちます。アプリケーションは、*カナリアリリース*と呼ばれるタイプの展開を使用するのが一般的です。このタイプの展開では、新バージョンにルーティングするユーザーの割合を徐々に増やして、トラフィックはゆっくりと新バージョンに切り換えられます。アプリケーションのバージョン間でクライアントを分割すると、新しいバージョンのコードをロールアウトする際にエラーが発生しても影響の範囲を制限するのに役立ちます。さらに一般的なのは、ブルーグリーン展開スタイルです。このスタイルでは、ユーザーは新しいバージョンの使用に切り替えられますが、展開が検証されている間、古いバージョンを引き続き使用できます。2つの異なるアプリケーションセットにクライアントを分割する理由が何であれ、NGINXなら、このsplit_clientsモジュールを提供するためシンプルに実行できます。

関連項目

split_clientsモジュールドキュメンテーション

3.2 GeoIPモジュールとデータベースの使用

問題

GeoIPデータベースをインストールし、NGINX内に埋め込まれた変数を有効にして、NGINXログ、プロキシされた要求、または要求ルーティング内のクライアントの物理的な場所を利用する必要があります。

解決法

公式のNGINXオープンソースパッケージリポジトリはNGINXのインストール時に第2章で構成された場合nginx-module-geoipという名前のパッケージを提供します。NGINX Plusパッケージリポジトリを使用する時、このパッケージはnginx-plus-module-geoipと名づけられます。ただし、NGINX Plusは、GeoIP2用の動的モジュールを提供しています。これは、HTTPと同様にNGINXストリームでも動作する、アップデートされたモジュールです。GeoIP2モジュールについては後ほど説明します。以下の例は、動的バージョンのNGINX GeoIPモジュールパッケージをインストールする方法と、GeoIP国と都市のデータベースをダウンロードする方法を示しています。オリジナルのGeoIPモジュールのデータベースの保守は終了しているので注意してください。

YUMパッケージマネージャーでNGINXオープンソースをインストール：

```
$ yum install nginx-module-geoip
```

APTパッケージマネージャーでNGINXオープンソースをインストール：

```
$ apt install nginx-module-geoip
```

YUMパッケージマネージャーでNGINX Plusをインストール：

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

APTパッケージマネージャーでNGINX Plusをインストール：

```
$ apt install nginx-plus-module-geoip
```

GeoIP国と都市のデータベースをダウンロードしてアンジップします：

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

このコマンドのセットは*geoip*ディレクトリを*/etc/nginx*ディレクトリに作成し、この新しいディレクトリに移動し、パッケージをダウンロードして解凍します。

国と都市のGeoIPデータベースがローカルディスクに保存し、NGINX GeoIPモジュールを用いてクライアントのIPアドレスに基づいて埋め込まれた変数を取得するよう指示することができます。

```
    load_module modules/ngx_http_geoip_module.so;

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

`load_module`ディレクティブはfilesystemのパスから動的にモジュールを読み込みます。`load_module`ディレクティブはメインコンテキストでのみ有効です。`geoip_country`ディレクティブはIPアドレスを国コードにマッピングするデータベースを含み、`http`コンテキストでのみ有効なGeoIP.datファイルへのパスを取ります。

GeoIP2モジュールは、動的モジュールとして利用でき、NGINXオープンソースのビルド時にコンパイルできます。NGINXオープンソースとGeoIP2モジュールのコンパイルについては、本書の範囲外です。以下の資料では、`nginx-plus-module-geoip2`モジュールのインストール方法について説明しています。

APTパッケージマネージャーを使用してGeoIP2用のNGINX Plus動的モジュールをインストールする

```
$ apt install nginx-plus-module-geoip2
```

YUMパッケージマネージャーを使用してGeoIP2用のNGINX Plus動的モジュールをインストールする

```
$ yum install nginx-plus-module-geoip2
```

GeoIP2用の動的モジュールをロードする：

```
load_module modules/ngx_http_geoip2_module.so;
load_module modules/ngx_stream_geoip2_module.so;

http {
  # ...
}
```

無料のMaxMind GeoLite2データベース（サインアップが必要です）をダウンロードします。次に、NGINXがこのデータベースを使用し、変数を介してGeo情報を公開するように設定します：

```
http {
  ...
  geoip2 /etc/maxmind-country.mmdb {
    auto_reload 5m;
```

```
      $geoip2_metadata_country_build metadata build_epoch;
      $geoip2_data_country_code default=US
        source=$variable_with_ip country iso_code;
      $geoip2_data_country_name country names en;
      }

      geoip2 /etc/maxmind-city.mmdb {
        $geoip2_data_city_name default=London city names en;
      }
      ....

      fastcgi_param COUNTRY_CODE $geoip2_data_country_code;
      fastcgi_param COUNTRY_NAME $geoip2_data_country_name;
      fastcgi_param CITY_NAME $geoip2_data_city_name;
      ....
    }

    stream {
      ...
      geoip2 /etc/maxmind-country.mmdb {
        $geoip2_data_country_code default=US
          source=$remote_addr country iso_code;
    }
    ...
  }
```

解説

この機能を使用するためには、NGINX GeoIPまたはGeoIP2モジュールをインストールし、ローカルにGeoIPの国と都市のデータベースがなければなりません。これらの前提条件のインストールと取得についてはこのセクションで説明したとおりです。

オリジナルのGeoIPでは、`geoip_country`と`geoip_city`ディレクティブは、このモジュールで予め用意された多数の変数を利用可能にします。`geoip_country`ディレクティブはクライアントのオリジン国を判断できる変数を有効にします。これらの変数には`$geoip_country_code`、`$geoip_ country_code3`、`$geoip_country_name`が含まれます。国コード変数は2文字の国コードを返し、末尾に3が付いた変数は3文字の国コードを返します。国名変数は国のフルネームを返します。

`geoip_city`ディレクティブはかなりの数の変数を有効にします。`geoip_city`ディレクティブは`geoip_country`ディレクティブとまったく同じ変数を有効にしますが、`$geoip_ city_country_code`、`$geoip_city_country_code3`、`$geoip_city_country_name`という異なる名前で呼ばれます。その他の変数、`$geoip_city`、`$geoip_lati tude`, `$geoip_ longitude`、`$geoip_city_continent_code`、`$geoip_postal_code`などは、すべて返す値を示しています。`$geoip_region`や`$geoip_region_name`は地域、領域、州、県、連邦領域などの説明を提供します。Region（地域）は2 文字のコードで、`regionname`（地域名）はフルネームです。`$geoip_area_code`は、米国でのみ有効

で、3 桁の電話市外局番を返します。

GeoIP2モジュールを使用する場合、geoip2ディレクティブは、同じ変数を使用しますが、変数のプレフィックスは、geoip_ではなく、geoip2_data_です。geoip2ディレクテブは、デフォルト、およびデータベースがMaxMindからリロードされる間隔も設定します。

これらの変数を使用してクライアントに関する情報を記録することができます。また、オプションで、この情報をヘッダーまたは変数としてアプリケーションに渡すか、NGINXを使用して特定の方法でトラフィックをルーティングできます。

関連項目

geoipモジュールドキュメンテーション

GeoIPアップデートGitHub

NGINX GeoIP2動的モジュールドキュメンテーション

GitHubでのGeoIP2のソースリポジトリとドキュメンテーション

3.3 国に基づくアクセス制限

問題

契約上の理由から、またはアプリケーションの要件のために、特定の国からのアクセスを制限する必要があります。

解決法

ブロックまたは許可する国コードを変数にマップします:

```
load_module modules/ngx_http_geoip2_module.so;
http {
  geoip2 /etc/maxmind-country.mmdb {
    auto_reload 5m;
    $geoip2_metadata_country_build metadata build_epoch;
    $geoip2_data_country_code default=US
      source=$variable_with_ip country iso_code;
    $geoip2_data_country_name country names en;
  }
  map $geoip2_data_country_code $country_access {
    "US" 0;
    default 1;
  }
  # ...
}
```

このマッピングは新しい変数$country_accessを1または0に設定します。クライアントのIPアドレスが、USからの場合、変数は0に設定されます。その他の国の場合には、すべて1に設定

されます。

では、serverブロック内でifステートメントを使用してUS以外の国からのアクセスを拒否してみましょう：

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

このifステートメントは$country_access変数が1に設定されている場合にはTrueと評価されます。Trueの場合、サーバーは403未認証を返します。それ以外の場合にはサーバーは通常通り機能します。そのため、このifブロックは、US以外の国からの人によるアクセスを拒否するためだけに存在します。

解説

これは、一部の国からのアクセスのみを許可する方法に関する簡単な例です。この例はニーズに合わせて応用できます。同じ方法をgeop2モジュールで利用できる任意の埋め込み変数に基づき許可またはブロックする場合に利用できます。

3.4 オリジナルクライアントを見つける

問題

NGINXサーバーの前にプロキシがあるため、オリジナルのクライアントIPアドレスを見つける必要があります。

解決法

geoip2_proxyディレクティブを使用してプロキシIPアドレス範囲を定義するため、geoip_proxy_recursiveディレクティブをオリジナルのIPを見つけるために使用します。

```
load_module "/usr/lib64/nginx/modules/ngx_http_geoip_module.so";
http {
  geoip2 /etc/maxmind-country.mmdb {
    auto_reload 5m;
    $geoip2_metadata_country_build metadata build_epoch;
    $geoip2_data_country_code default=US
      source=$variable_with_ip country iso_code;
    $geoip2_data_country_name country names en;
  }
  geoip2_proxy 10.0.16.0/26;
  geoip2_proxy_recursive on;
# ...
```

```
}
```

geoip2_proxyディレクティブはクラスレスドメイン間ルーティング（CIDR）範囲を定義します。この範囲内でプロキシサーバーは稼働し、X-Forwarded-Forヘッダーを利用してクライアントIPアドレスを見つけるようにNGINXに指示します。geoip2_proxy_recursiveディレクティブは、最後の既知のクライアントIPのX-Forwarded-Forを再帰的に調べるようにNGINXに指示します。

Forwardedとついたヘッダーはプロキシされた要求のプロキシ情報を追加するための標準ヘッダーになりました。NGINX GeoIP2 モジュールに使用されるヘッダーはX-Forwarded-Forで記述時に指定することはできません。X-Forwarded-Forは公式の標準ではありませんが、広く使用され、認められ、多くのプロキシにより設定されている形式です。

解説

NGINXの前でプロキシを使用している場合、NGINXはクライアントではなくプロキシのIPアドレスを取得することがあります。この場合、geoip2_proxyディレクティブを使用して、指定範囲から接続が開かれたときにX-Forwarded-Forヘッダーを使うようにNGINXに指示できます。geoip2_proxyディレクティブはアドレスまたはCIDR範囲を取ります。NGINXの前にトラフィックを渡す複数のプロキシがある場合、geoip2_proxy_recursiveディレクティブを使用して、X-Forwarded-Forアドレスを再帰的に検索し、発信元のクライアントを見つけることができます。NGINXの前でAmazon Web Services Elastic Load Balancing（AWS ELB）、Google Cloud Platformのロードバランサー、またはMicrosoft Azureのロードバランサーなどを利用する場合はこうしたディレクティブの使用をお勧めします。

3.5 接続制限

問題

クライアントのIPアドレスなどの事前定義されたキーに基づいて、接続数を制限する必要があります。

解決法

接続メトリクスを保持する共有メモリーゾーンを構築し、limit_connディレクティブを使用して、接続を制限します：

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

```
    }
  }
```

この構成により、limitbyaddrという名前の共有メモリーゾーンが作成されます。使用される事前定義されたキーは、バイナリ形式のクライアントのIPアドレスです。共有メモリーゾーンのサイズは10MBに設定されています。limit_connディレクティブは2つのパラメータを使用します：limit_conn_zone名と許可される接続数です。 limit_conn_statusはステータス429で接続が制限されている時「too many requests」と表示する応答を設定します。limit_connとlimit_conn_statusディレクティブはhttp、server、locationコンテキストで有効です。

解説

キーに基づく接続数制限は、侵害からの防御やすべてのクライアント間での公平なリソースの共有目的で使用することができます。事前定義されたキーには注意を払うことが重要です。前述の例のようにIPアドレスを使用すると同じIPから発信された同じネットワーク上に多くのユーザーがいる場合に危険になる可能性があります。これには、Network Address Translation（NAT）の背後に多数のユーザーがいる場合などが該当します。 この場合、クライアント全体が制限されます。limit_conn_zoneディレクティブはhttpコンテキストでのみ有効です。制限に使用する文字列を作成するために、httpコンテキスト内でNGINXが利用できる任意の数の変数を利用することができます。セッションcookieなど、アプリケーションレベルでユーザーを識別できる変数を利用することが、ユースケースによってはよりクリーンなソリューションになるかもしれません。limit_conn_statusのデフォルトは503、「サービス利用不可」です。サービスが利用可能であるため、429の使用が好まれるかもしれません。500レベルの応答はサーバーエラー、400レベルの応答はクライアントエラーを示すためです。

この制限についてテストをすることは少々難しいかもしれません。テスト用の代替環境でライブのトラフィックをシミュレートするのは難しい場合が多いためです。この場合、limit_conn_dry_runディレクティブを有効にして、アクセスログの中で変数$limit_conn_statusを使用します。$limit_conn_status変数はPASSED、DELAYED、REJECTED、DELAYED_DRY_RUN、REJECTED_DRY_RUNのいずれかを選択します。ドライランを有効にすると、ライブトラフィックのログを分析し、制限を超えるリクエストを拒否する前に必要に応じて制限を微調整できます。

3.6 レート制限

問題

クライアントのIPアドレスなど、事前定義されたキーによってリクエストのレートを制限する必要があります。

解決法

レート制限モジュールを利用して、リクエストのレートを制限します：

```
http {
  limit_req_zone $binary_remote_addr
```

```
    zone=limitbyaddr:10m rate=3r/s;
  limit_req_status 429;
  # ...
  server {
    # ...
      limit_req zone=limitbyaddr;
    # ...
  }
}
```

この設定例では、limitbyaddrという名前の共有メモリーゾーンを作成します。使用される事前定義されたキーは、バイナリ形式のクライアントのIPアドレスです。共有メモリーゾーンのサイズは10MBに設定されています。ゾーンは、キーワード引数を使用してレートを設定します。limit_reqはディレクティブは必須のキーワード引数を取ります：zone。zoneは使用する共有メモリ要求制限ゾーンにディレクティブを指示します。レートを超える要求は、limit_req_statusディレクティブで定義されているように429 HTTPコードが返されます。これは、問題が実際にはクライアントにある場合に、サーバーに問題があるように示唆します。

limit_reqディレクティブにオプションのキーワード引数を使用して、2段階のレート制限を有効にします。

```
server {
  location / {
    limit_req zone=limitbyaddr burst=12 delay=9;
  }
}
```

場合によっては、クライアントは一度に多くの要求を行う必要があり、その後、さらに要求を行う前に一定期間レートを下げます。キーワード引数burstを使用して、クライアントに要求を却下せずに、レート制限を超過することを許可することができます。 レートを超過した要求は、設定された値までレート制限に一致するように処理が遅延されます。以下のキーワード引数のセットはこの振る舞いを変更します：delayとnodelayです。 nodelay引数は値を取りません。クライアントがバースト可能な値を一度に消費できるようにしますが、レート制限を満たすだけの時間が経過するまで、すべての要求は拒否されます。この例で、nodelayを使用した場合には、クライアントは最初の1秒間に12の要求を消費できますが、別の要求を行うには、初めの要求から4秒待つ必要があります。delayキーワード引数はスロットルなしで事前に作成できる要求の数を定義します。このケースでは、クライアントは遅延なしで9つの要求を前もって行うことができ、次の3つは抑制され、4秒以内に発生した追加の要求は拒否されます。

解説

レート制限モジュールは、すべての人に質の高いサービスを提供しながら、不正で急速な要求から保護するために非常に強力です。要求のレートを制限する理由はたくさんありますが、その1つがセキュリティです。ログインページに非常に厳しい制限を設けることで、ブルートフォース攻撃を拒否できます。すべての要求に適切な制限を設定して、アプリケーションでサービス拒否を発生させたり、リソースを浪費したりしようとする悪意のあるユーザーの計画を無効にすることができます。

レート制限モジュールの構成は、レシピ3.5で説明した接続制限モジュールとよく似ており、その懸念事項の多くがここでも当てはまります。1秒あたりの要求数または1分あたりの要求数で制限するレートを指定できます。レート制限に達したら、インシデントがログされます。この例では使用されていないディレクティブもあります。`limit_req_log_level`はデフォルトで`error`になりますが、`info`、`notice`、`warn`に設定することもできます。NGINX Plusでは、レート制限はクラスタ上で動作します。 (ゾーンの同期の例はこちら：レシピ 12.5)。

この制限についてテストをすることは少々難しいかもしれません。テスト用の代替環境でライブのトラフィックをシミュレートするのは難しい場合が多いためです。この場合、`limit_conn_dry_run`ディレクティブを有効にして、アクセスログの中で変数`$limit_conn_status`を使用します。`$limit_conn_status`変数は`PASSED`、`REJECTED`、`REJECTED_DRY_RUN`のいずれかを選択します。ドライランを有効にすると、ライブトラフィックのログを分析し、制限を超えるリクエストを拒否する前に必要に応じて制限を微調整できます。

3.7 帯域幅 制限

問題

クライアントごとのアセットダウンロード帯域幅を制限する必要があります。

解決法

NGINXの`limit_rate`と`limit_rate_after`ディレクティブを使用して、クライアントへの応答の帯域幅を制限します：

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

このロケーションブロックの構成では、プレフィックス*download*でURIを制限し、応答がクライアントに提供される速度が10MB以降、1MB /秒の速度に制限されるように指定されています。帯域幅の制限は接続ごとであるため、必要に応じて、接続の制限と帯域幅の制限を設定することをお勧めします。

解説

設定の接続の帯域幅を制限すると、NGINXは指定した方法ですべてのクライアント間でアップロードの帯域幅を共有するようにします。これら2つのディレクティブです：`limit_rate_after`および`limit_rate`ディレクティブは、`http`、`server`、`location`といったほぼすべてのコンテキストで設定でき、`if`がlocation内にある場合には、`if`も使用できます。`limit_rate`ディレクティブは`limit_rate_after`と同じコンテキストで適用されますが、`$limit_rate`という名前の変数でも設定することができます。

`limit_rate_after`ディレクティブは指定された量のデータが転送されるまで接続をレート制限しないように指定します。`limit_rate`ディレクティブは、デフォルトで、特定のコンテキスト

のレート制限を1秒あたりのバイト数で指定します。しかし、メガバイトに mまたはギガバイトにはgも指定できます。どちらのディレクティブもデフォルトの値は 0です。値0とは、ダウンロードレートをまったく制限しないことを意味します。このモジュールを使用すると、クライアントのレート制限をプログラムで変更できます。

第4章

大幅にスケーラブルなコンテンツキャッシング

4.0 はじめに

キャッシングは、将来再び提供される要求への応答を保存することにより、コンテンツの提供を速くします。NGINXは、キャッシュから提供することで、反復作業をオフロードしてアップストリームサーバーの負荷を軽減します。キャッシュ機能により、パフォーマンスが向上し、負荷が減ります。つまり、より少ないリソースでより高速なサービス提供が可能になります。また、リソース提供にかかる時間と帯域幅が減ります。

戦略的な場所でのキャッシュサーバーのスケーリングと分散は、ユーザーエクスペリエンスに大きなプラスの効果が生じる可能性があります。最高のパフォーマンスを得るには、消費者の近くでコンテンツをホストするのが最適です。また、ユーザーの近くにコンテンツをキャッシュすることもできます。これがコンテンツ配信ネットワーク、CDNのパターンです。NGINXなら、NGINXを配置できる場所ならどこででもコンテンツをキャッシュでき、効果的に自身のCDNを作成できるようになります。NGINXのキャッシュでパッシブ（受動的）にキャッシュできるため、アップストリームで障害発生時にはキャッシュされた応答を返すこともできます。キャッシュ機能は`http`コンテキストでのみ利用できます。この章では、NGINXのキャッシュ機能とコンテンツ配信機能について説明します。

4.1 ゾーンキャッシュ

問題

コンテンツをキャッシュし、キャッシュの保存場所を定義する必要があります。

解決法

`proxy_cache_path`ディレクティブを使用して、共有メモリーキャッシュゾーンを定義し、locationをコンテンツに使用します：

```
proxy_cache_path /var/nginx/cache
                 keys_zone=main_content:60m
                 levels=1:2
                 inactive=3h
                 max_size=20g
                 min_free=500m;
proxy_cache CACHE;
```

キャッシュ定義の例では、ファイルシステムにキャッシュされた応答用のディレクトリを*/var/nginx/cache*に作成し、`main_content`という名前の共有メモリースペース（60 MBのメモリー容量）を作成します。この例では、ディレクトリ構造レベルを設定し、キャッシュされた応答が3時間以内に要求されなかった場合に解放されるように指定して、キャッシュの最大サイズ20 GBを定義します。`min_free`パラメーは、キャッシュされたリソースを解放する前に、`max_size`からどれだけのディスク領域を空けておくかをNGINXに指示します。`proxy_cache`ディレクティブは特定のコンテキストにキャッシュゾーンを使用するよう指示します。`proxy_cache_path`は`http`コンテキストで有効で、`proxy_cache`ディレクティブは`http`、`server`、`location`コンテキストで有効です。

解説

NGINX でキャッシュを構成するには、使用するパスとゾーンを宣言する必要があります。NGINXのキャッシュゾーンは`proxy_cache_path`ディレクティブで作成されます。`proxy_cache_path`ディレクティブはキャッシュされた情報を保存する場所と、アクティブなキーと応答メタデータを保存する共有メモリースペースを指定します。このディレクティブのオプションのパラメータはキャッシュの保持とアクセスにより細かな制御を提供します。

`levels`パラメータはディレクトリ構造の作成方法を定義します。値は、コロンで区切られた値リストで、サブディレクトリ名の長さを定義します。より深い構造にすると、1つのディレクトリ内に表示されるキャッシュされたファイルの数が多くなりすぎなくなります。次に、NGINXは、キャッシュキーをファイルパスとして使用し、`levels`値に基づいてディレクトリを分割して、提供されたファイル構造に結果を保存します。

`inactive`パラメータは、キャッシュされたアイテムが最後に使用された時点からホストされる時間の長さを制御できるようにします。キャッシュのサイズも`max_size`パラメータを使って構成可能です。その他のパラメータは、ディスクにキャッシュされたファイルから共有メモリーゾーンにキャッシュキーをロードするキャッシュロードプロセス、およびその他の多くのオプションに関連しています。`proxy_cache_path`ディレクティブの詳細については、以下の「関連項目」セクションでドキュメンテーションへのリンクを探してください。

関連項目

`proxy_cache_path`ドキュメンテーション

4.2 キャッシュハッシュキーのキャッシュ

問題

コンテンツのキャッシュ方法と取得の方法を制御する必要があります。

解決法

proxy_cachekeyディレクティブと変数を使ってキャッシュヒットするかしないかを構成するものを定義できます。

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

このキャッシュハッシュキーは、ホストと要求されているURI、およびユーザーを定義するcookieに基づいてページをキャッシュするようにNGINXに指示します。こうすることで、別のユーザー向けに生成されたコンテンツを提供せずに動的ページをキャッシュできます。

解説

ほとんどのユースケースに適したデフォルトのproxy_cache_keyは"$scheme$proxy_host$request_uri"です。使用される変数には、スキーム(httpまたはhttps)、リクエストが送信されるproxy_host、および要求されたURIが含まれます。これは全体でNGINXがプロキシを行うリクエストのURLを表します。要求引数、ヘッダー、セッション識別子など、アプリケーションごとに一意の要求を定義するその他多数の要因があります。それに対し、独自のハッシュキーを作成したいと思われるかもしれません。[1]

優れたハッシュキーの選択は非常に重要です。これはアプリケーションに対する理解に基づき、熟慮を重ねるべきでしょう。静的コンテンツのキャッシュキーの選択は、通常、非常に簡単で、ホスト名とURIを使用するだけで十分です。ダッシュボードアプリケーションのページなど、動的なコンテンツのキャッシュキーを選択するには、ユーザーがアプリケーションを操作する方法と、ユーザーエクスペリエンス間の差異に関する知識をもつ必要があります。動的なコンテンツをキャッシュする場合、CookieやJWTトークンのようなセッション識別子を使用すると非常に便利です。セキュリティ上の懸念から、コンテキストを完全に理解せずに、キャッシュされたデータをあるユーザーから別のユーザーに提供したくない場合があります。proxy_cache_keyディレクティブはキャッシュキーのハッシュされる文字列を構成します。proxy_cache_keyはhttp、server、locationブロックのコンテキストで設定でき、要求がハッシュされる方法を柔軟に制御できるようになります。

1 NGINXに公開されるテキストまたは変数の任意の組み合わせを使用して、キャッシュキーを形成できます。変数の一覧はNGINXで利用できます。

4.3 キャッシュロッキング

問題

キャッシュが更新されるリソースに対する同時リクエストをNGINXがどのように処理するかを制御する必要があります。

解決法

proxy_cache_lockディレクティブを使用して、一度に1つの要求しかキャッシュに書き込めないようにします。後続の要求は、応答がキャッシュに書き込まれ、そこからサービスが提供されるのを待ちます。

```
proxy_cache_lock on;
proxy_cache_lock_age 10s;
proxy_cache_lock_timeout 3s;
```

解説

コンテンツに対する要求が送信中で、キャッシュに書き込まれている途中である場合、同じコンテンツに対する後続の要求をプロキシしないようにできます。proxy_cache_lockディレクティブは、現在入力されている、キャッシュされたリソース宛ての要求を保持するようにNGINXに指示します。キャッシュにデータを入力しているプロキシされた要求は、別の要求がリソースにデータを入力しようとするまでの時間に制限があります。これは、proxy_cache_lock_ageディレクティブで定義され、デフォルトは5秒です。NGINXは、指定された時間待機していた要求をプロキシサーバーに渡すよう許可することもできます。プロキシサーバーは、proxy_cache_lock_timeoutディレクティブを使用してキャッシュにデータを入力しようとしません。これもデフォルトは5秒です。NGINX は、指定された時間待機していた要求をプロキシサーバーに渡すよう許可することもできます。プロキシサーバーは、proxy_cache_lock_timeout ディレクティブを使用してキャッシュにデータを入力しようとしません。これもデフォルトは 5 秒です。この2 つのディレクティブの違いは以下の例を考えるとわかりやすいでしょう。 proxy_cache_lock_age は「君は遅すぎるので、私が君のためにキャッシュを入力してあげよう」と言い、proxy_cache_lock_timeout は「君は僕を長く待たせすぎる。僕が最初に自分が欲しい物を取るから、君は君のペースでキャッシュすればいいよ。」と答えます。

4.4 古いキャッシュの使用

問題

アップストリームサーバーを利用できない場合に、期限切れのキャッシュエントリを送信する必要があります。

解決法

NGINXが古いキャッシュを使用する条件を定義するパラメータ値を指定して、proxy_cache_use_staleディレクティブを使用します：

```
proxy_cache_use_stale error timeout invalid_header updating http_500
  http_502 http_503 http_504
  http_403 http_404 http_429;
```

解説

NGINXは、アプリケーションが正しく応答しないときに、キャッシュされた古いリソースを提供できる便利な機能があります。この機能によって、バックエンドがアクセスできない状態でも、エンドユーザーにはWebサービスが機能しているように見えます。また、古いキャッシュからサービスを提供することで、問題が発生した場合のバックエンドサーバーへの負荷が軽減されるので、エンジニアリングチームが問題をデバッグする余裕が生まれます。

この構成では、アップストリーム要求がタイムアウトになる、invalid_headerエラーが返される、または400レベルや500レベルの応答コードが返されると、NGINXは古いキャッシュリソースを使用します。errorパラメータとupdatingパラメータは少し特殊です。errorパラメータは、アップストリームサーバーを選択できない場合に古いキャッシュの使用を許可します。updatingパラメータは、キャッシュが新しいコンテンツで更新されている間、キャッシュされた古いリソースを使用するようにNGINXに指示します。

関連項目

NGINX Use Stale Cacheディレクティブドキュメンテーション

4.5 キャッシュバイパス

問題

キャッシュをバイパスする機能が必要です。

解決法

proxy_cache_bypassディレクティブと、空/ ゼロではない値を使用します。これを動的に行う1つの方法は、キャッシュしたくないロケーションブロック内の変数を空の文字列以外または0以外に設定することです。

```
proxy_cache_bypass $http_cache_bypass;
```

この構成は、cache_bypassという名前のHTTP要求ヘッダーが0以外の値に設定されている場合、キャッシュをバイパスするようにNGINXに指示します。この例では、ヘッダーを変数として使用して、キャッシュをバイパスする必要があるかどうかを判断します。クライアントは、要求に対してこのヘッダーを具体的に設定する必要があります。

解説

要求をキャッシュしない必要が生じるシナリオは多々あります。このため、NGINXは proxy_cache_bypassディレクティブを提供します。値が空でないかゼロ以外の場合、要求はキャッシュから提供されるのではなく、アップストリームサーバーに送信されます。キャッシュをバイパスするためのさまざまなニーズとシナリオは、アプリケーションのユースケースによって決まります。キャッシュをバイパスする手法は、要求ヘッダーまたは応答ヘッダーを使用するのと同じくらい単純な場合もあれば、複数のマップブロックが連携して機能するのと同じくらい複雑な場合もあります。

キャッシュをバイパスするよう希望する理由は、トラブルシューティングやバグ修正などさまざまです。キャッシュされたページを一貫して取得して提供している場合、またはキャッシュキーがユーザー識別子に固有である場合、問題の再現が難しくなります。キャッシュをバイパスできる機能は不可欠です。オプションとして、特定のcookie、ヘッダー、または要求引数が設定されたときにキャッシュをバイパスすることが考えられますが、これに限定されません。proxy_cache をオフに設定することで、ロケーションブロックなどの特定のコンテキストにキャッシュを完全にオフにすることもできます。

4.6 NGINX Plus でのキャッシュパージ

問題

キャッシュからのオブジェクトを無効にする必要があります。

解決法

クライアント側キャッシュ制御ヘッダーを使用します：

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

この例では、PURGEメソッドで要求された場合に特定のオブジェクトのキャッシュがパージされます。以下は *main.js* という名前のファイルのキャッシュをパージするcurl例です。

```
$ curl -X PURGE http://www.example.com/main.js
```

解説

静的ファイルを処理する一般的な方法は、ファイル名にファイルのハッシュを入れることです。これにより、新しいコードとコンテンツをロールアウトする際に、CDN はそれを新しいファイルとして認識します。URIが変更されたためです。ただし、これは、このモデルに適合しないキャッシュキーを設定した動的コンテンツでは正確に機能しません。すべてのキャッシュシナリオで、キャッシュをパージする方法が必要です。

NGINX Plusはキャッシュ応答をパージするシンプルな方法を提供します。`proxy_cache_purge`ディレクティブは、ゼロまたは空ではない値を渡すと要求に適合するキャッシュされたアイテムをパージします。パージを設定する簡単な方法は、`PURGE`の要求方法をマッピングすることです。しかし、これを`geoip`モジュールまたは単純な認証と組み合わせて使用して、貴重なキャッシュアイテムを誰もパージできないようにしたいと思うことがあるかもしれません。NGINX では、* の使用も許可されています。これにより、共通のURIプレフィックスに一致するキャッシュアイテムがパージされます。ワイルドカードを使用するには、`proxy_cache_path`ディレクティブを`purger=on`引数と一緒に構成する必要があります。

関連項目

NGINXキャッシュパージの例

4.7 キャッシュスライシング

問題

リソースをフラグメントに分割して、キャッシュ効率を改善する必要があります。

解決法

NGINXの`slice`ディレクティブとその埋め込み変数を使用して、キャッシュ結果をフラグメントに分割します。

```
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;
  }
}
```

解説

この構成は、キャッシュゾーンを定義し、サーバーに対して有効にします。sliceディレクティブはその後、NGINX に応答を1 MB セグメントにスライスするよう指示するために使用されます。キャッシュされたリソースは、proxy_cache_keyディレクティブに従って保存されます。slice_rangeという名前の埋め込み変数の使用に注意してください。同じ変数が発信元に要求する時にヘッダーとして使用され、1.0はバイト範囲要求をサポートしていないため、その要求のHTTPバージョンはHTTP /1.1にアップグレードされます。キャッシュの有効性は、応答コード200または206に1時間設定され、場所と発信元が定義されます。

Cache Sliceモジュールは、HTML5ビデオを配信するために開発されました。このモジュールは、バイト範囲の要求を使用して、コンテンツをブラウザに疑似ストリーミングします。デフォルトでは、NGINXはバイト範囲要求をキャッシュから提供することができます。キャッシュされていないコンテンツに対してバイト範囲の要求が行われると、NGINXはファイル全体を発信元に要求します。CacheSliceモジュールを使用する場合、NGINXは発信元に必要なセグメントのみを要求します。ファイル全体を含め、スライスサイズよりも大きい範囲要求は、必要な各セグメントのサブリクエストをトリガーし、それらのセグメントがキャッシュされます。すべてのセグメントがキャッシュされると、レスポンスが見立てられ、クライアントに送信されます。NGINX はより効率的にキャッシュを行い、範囲指定して要求されたコンテンツを提供することができます。

CacheSliceモジュールは、変更が発生しない大きなリソースでのみ使用されるべきです。NGINXはオリジンからセグメントを受領する度に、ETagを確認します。オリジンのETagが変更されると、キャッシュが無効になるため、NGINXはセグメントのキャッシュの読み込みを中断します。コンテンツが変更され、ファイルが小さくなった場合、またはオリジンがキャッシュフィル処理中の負荷の急増に対処できる場合、レシピ4.3で説明されているcache_lockディレクティブを使うことをお勧めします。キャッシュスライシング手法については、「関連項目」セクションにあるブログを参照してください。

関連項目

“Smart and Efficient Byte-Range Caching with NGINX & NGINX Plus”

第5章
プログラマビリティと自動化

5.0 はじめに

プログラマビリティとは、プログラミングを通じて何かと対話する能力を指します。NGINX PlusのAPIはこの能力を提供します：HTTPのインターフェースを介してNGINX Plusの構成や動作と対話する機能です。このAPIは、HTTP要求を介してアップストリームサーバーを追加または削除することにより、NGINX Plusを再構成する機能を提供します。NGINX Plusのキーバリューストア機能により、一段上のレベルの動的構成が可能になります。HTTP呼び出しを利用して、NGINX Plusがトラフィックを動的にルーティングまたは制御するために使用できる情報を注入できます。本章では、NGINX Plus APIと、このAPIによって公開されるキーバリューストアモジュールについて説明します。

構成管理ツールは、サーバーのインストールと構成を自動化します。これは、クラウドの時代において非常に貴重なユーティリティです大規模なWebアプリケーションのエンジニアは、サーバーを手動で構成する必要がなくなり、代わりに、利用可能な多くの構成管理ツールのいずれかを使用できます。こうしたツールを使用すると、エンジニアは構成とコードを1回記述するだけで、同じ構成の多くのサーバーを繰り返し可能でテスト可能なモジュール方式で作成できます。本章では、利用可能な最も一般的な構成管理ツールの一部と、それらを使用してNGINXをインストールし、基本構成をテンプレート化する方法について説明します。これらの例は非常に基本的なものですが、各プラットフォームでNGINXサーバーの使用を開始する方法を示します

5.1 NGINX Plus API

問題

動的な環境で、NGINX Plusを自動で再構成する必要があります。

解決法

NGINXPlus APIを構成して、API呼び出しによるサーバーの追加と削除を有効にします。

```
upstream backend {
  zone http_backend 64k;
}
server {
  # ...
  # enable /api/ location with appropriate access
  # control in order to make use of NGINX Plus API
  location /api {
    # Set write=off for read only mode, recommended
    api write=on;
    # Directives limiting access to the API
    # See chapter 7
  }
  # enable NGINX Plus Dashboard; requires /api/ location to be
  # enabled and appropriate access control for remote access
  location = /dashboard.html {
    root /usr/share/nginx/html;
  }
}
```

このNGINX Plus構成は、共有メモリーゾーンを持つアップストリームサーバーを作成し、/apiロケーションブロックでAPIを有効にし、NGINX Plusダッシュボードのロケーションを提供します。

APIを利用して、サーバーがオンラインになった時にサーバーを追加できます。

```
$ curl -X POST -d '{"server":"172.17.0.3"}' \
  'http://nginx.local/api/9/http/upstreams/backend/servers/'
{
  "id":0,
  "server":"172.17.0.3:80",
  "weight":1,
  "max_conns":0,
  "max_fails":1,
  "fail_timeout":"10s",
  "slow_start":"0s",
  "route":"",
  "backup":false,
  "down":false
}
```

この例のcurl呼び出しは、バックエンドのアップストリーム構成に新しいサーバーを追加するようにNGINX Plusに要求します。HTTPメソッドはPOSTで、JSONオブジェクトが本文として渡され、JSON応答が返されます。JSON応答は、サーバーオブジェクトの構成を示します。新しいidが生成され、他の構成設定にはデフォルト値が設定されていることに注意してください。

NGINX Plus APIはRESTfulです。そのため、要求URIにパラメータがあります。

URIの形式は以下の通りです：

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

NGINX Plus APIを利用して、アップストリームプール内のサーバーを一覧表示できます：

```
$ curl 'http://nginx.local/api/9/http/upstreams/backend/servers/'
[
  {
    "id":0,
    "server":"172.17.0.3:80",
    "weight":1,
    "max_conns":0,
    "max_fails":1,
    "fail_timeout":"10s",
    "slow_start":"0s",
    "route":"",
    "backup":false,
    "down":false
  }
]
```

この例のcurl呼び出しは、backendという名前のアップストリームプール内のすべてのサーバーを一覧表示するようにNGINX Plusに要求します。現在、APIへの前回のcurl呼び出しで追加したサーバーは1つだけです。要求は、サーバーの構成可能なすべてのオプションを含むアップストリームサーバーオブジェクトを返します。

NGINX Plus APIを使用して、以下のコードのように、アップストリームサーバーから接続をドレインし、アップストリームプールから適切に削除できるように準備します。 以下に接続をドレインするためのAPI要求を示しますが、接続のドレインに関する詳細はレシピ2.8を参照してください。

```
$ curl -X PATCH -d '{"drain":true}' \
  'http://nginx.local/api/9/http/upstreams/backend/servers/0'
{
  "id":0,
  "server":"172.17.0.3:80",
  "weight":1,
  "max_conns":0,
  "max_fails":1,
  "fail_timeout":"10s",
  "slow_start":"0s",
  "route":"",
  "backup":false,
  "down":false,
  "drain":true
}
```

このcurlでは、HTTPメソッドとしてPATCHを指定し、サーバーへの接続をドレインするように指示するJSON本文を渡し、URIに対象のサーバーIDを指定します。 前のcurlコマンドでアップストリームプール内のサーバーを一覧表示して、サーバーのIDを確認したことを想定しています。

NGINX Plusは接続のドレインを開始します。このプロセスは、アプリケーションのセッションと同じくらい長くかかることがあります。ドレインを開始したサーバーによって提供されているアクティブな接続の数を確認するには、次の呼び出しを使用して、ドレインされているサーバーの情報を確認します：

```
$ curl 'http://nginx.local/api/9/http/upstreams/backend'
{
  "zone" : "http_backend",
  "keepalive" : 0,
  "peers" : [
    {
      "backup" : false,
      "id" : 0,
      "unavail" : 0,
      "name" : "172.17.0.3",
      "requests" : 0,
      "received" : 0,
      "state" : "draining",
      "server" : "172.17.0.3:80",
      "active" : 0,
      "weight" : 1,
      "fails" : 0,
      "sent" : 0,
      "responses" : {
        "4xx" : 0,
        "total" : 0,
        "3xx" : 0,
        "5xx" : 0,
        "2xx" : 0,
        "1xx" : 0
      },
      "health_checks" : {
      "checks" : 0,
      "unhealthy" : 0,
      "fails" : 0
      },
      "downtime" : 0
      }
  ],
  "zombies" : 0
}
```

すべての接続がドレインされたら、NGINX Plus APIを利用して、サーバーをアップストリームプールから完全に削除します:

```
$ curl -X DELETE \
  'http://nginx.local/api/9/http/upstreams/backend/servers/0'
[]
```

curlコマンドはDELETEメソッドの要求をサーバーの状態を更新する際と同じURIに実行します。DELETEメソッドは、サーバーを削除するようNGINXに指示します。このAPI呼び出しは、プールに残っているすべてのサーバーとそのIDを返します。今、私たちは空のプールから始めて、APIを介してサーバーを1つだけ追加し、それをドレインして削除したため、また空のプールだけが残りました。

解説

NGINX Plus専用のAPIを使用することで、動的アプリケーションサーバーは自動的に自身をNGINX構成に追加および削除できるようになります。サーバーがオンラインになると、プールに自分自身を登録することができ、NGINXはサーバーへ通信の送信を開始します。サーバを削除する必要がある場合、NGINX Plusはサーバの接続をドレインし、アップストリームプールからサーバを安全に削除できるように動作させることが可能です。これにより、インフラストラクチャは、オートメーションにより人間の介入なしにスケールインおよびスケールアウトできます。

関連項目

NGINX Plus REST API ドキュメンテーション

5.2 NGINX Plusでのキーバリューストアの使用

問題

アプリケーションからの入力に基づいて動的なトラフィック管理の決定を行うには、NGINX Plusが必要です。

解決法

このセクションでは、動的ブロックリストの例をトラフィック管理の決定基準として使用します。

クラスタ対応のキーバリューストアとAPIをセットアップしてから、キーと値を追加します:

```
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;
    }
  }
```

このNGINX Plus構成はkeyval_zoneディレクティブを使用してキーバリューストアをblocklistという名前の共有メモリーゾーンに構築し、メモリ制限を1 MBに設定します。次に、keyvalディレクティブはキーの値をマップし、最初のパラメータ$remote_addrをゾーンの$blockedという名前の新しい変数に一致させます。次に、この新しい変数を使用して、NGINX Plusが要求を処理するか、403 Forbiddenコードを返すかを決定します。

この構成でNGINX Plusサーバーを起動した後、ローカルマシンへcurlすると、200 OK応答を受け取ることを期待できます：

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

次に、ローカルマシンのIPアドレスを値1のキーバリューストアに追加します：

```
$ curl -X POST -d '{"127.0.0.1":"1"}' \
  'http://127.0.0.1/api/9/http/keyvals/blocklist'
```

このcurlコマンドはblocklist共有メモリーゾーンに送信されるキーバリューのオブジェクトを含むJSONオブジェクトを使用してHTTP POST要求を送信します。キー値ストアのAPI URIの形式は次のとおりです：

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

これで、ローカルマシンのIPアドレスが値1のblocklistという名前のキーバリューゾーンに追加されます。次の要求では、NGINX Plusはキーバリューゾーンで$remote_addrをルックアップし、エントリを見つけて、その値を変数$blockedにマップします。この変数は次にifステートメントで評価されます。変数に値がある場合、ifはTrueと評価され、NGINX Plusは403 Forbidden戻りコードを返します：

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

PATCHメソッド要求を作成して、キーを更新または削除できます：

```
$ curl -X PATCH -d '{"127.0.0.1":null}' \
  'http://127.0.0.1/api/9/http/keyvals/blocklist'
```

NGINX Plusは値がnullの場合にはキーを削除し、要求には200 OKが再び返されます。

解説

NGINX Plus独自の機能であるキーバリューストアを使用すると、アプリケーションでNGINX Plusに情報を注入できます。上述の例では、動的ブロックリストの作成に`$remote_addr`が使用されます。NGINX Plusが変数として持つ可能性のある任意のキー（セッションcookieなど）をキーバリューストアに入力し、NGINX Plusに外部の値を提供できます。NGINX Plus R16では、キーバリューストアがクラスタ対応になりました。つまり、キー値の更新を1つのNGINX Plusサーバーにのみ提供するだけで、すべてのサーバーが情報を受け取ります。

NGINX Plus R19ではキーバリューストアで`type`パラメータが有効になり、特定のタイプのキーのインデックス作成が可能になりました。デフォルトで、`type`は`string`で、`ip`や`prefix`がオプションとなります。`string`タイプはインデックスを作成せず、すべてのキー要求は完全に一致する必要があります。`prefix`タイプではprefixで指定した文字列に対し部分的に一致する必要があります。`ip`タイプを使用すると、CIDR表記を使用できます。ここで使用した例では、type=ipをゾーンのパラメータとして指定した場合、RFC1918プライベート範囲ブロック全体をブロックする`192.168.0.0/16`など、ブロックするCIDR範囲全体を指定できます。または、localhostの場合は`127.0.0.1/32`で、例に示されているのと同じ効果が得られます。

関連項目

ブログ（英語）:“Dynamic Bandwidth Limits Using the NGINX Plus Key-Value Store”

5.3 NJSモジュールを使ったNGINX内でのJavaScript機能の公開

問題

要求または応答に対してカスタムロジックを実行するためにNGINXが必要です。

解決法

NGINXのNGINX JavaScript（njs）モジュールをインストールすることで、JavaScriptを使用できるようになります。以下のパッケージのインストール手順は、第1章で示したように、お使いのLinuxディストリビューションにNGINXの公式リポジトリを追加していることを前提としています。

APTパッケージマネージャーでNGINXオープンソースをインストール：

```
$ apt install nginx-module-njs
```

APTパッケージマネージャーでのNGINX Plusをインストール：

```
$ apt install nginx-plus-module-njs
```

YUMパッケージマネージャーでのNGINXオープンソースをインストール：

```
$ yum install nginx-module-njs
```

YUMパッケージマネージャーでのNGINX Plusをインストール：

```
$ yum install nginx-plus-module-njs
```

NGINX構成内にJavaScriptファイル用のディレクトリがまだない場合は作成します：

```
$ mkdir -p /etc/nginx/njs
```

/etc/nginx/njs/jwt.jsという名前のJavaScriptファイルを、以下の内容で作成します：

```
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}
```

このJavaScriptの例では、JSON Webトークン（JWT）をデコードする関数を定義しています。さらに、JWTデコーダを使用してJWT内の特定のキーを返す2つの関数が定義されています。これらの関数は、NGINXで利用できるようにエクスポートされ、JWTにある共通のキー、つまりsubjectとissuerを返します。コードの`.slice(7)`の部分は、`Authorization`ヘッダー値の最初の7文字を削除するためのものです。JWTの出現により、タイプ値は`Bearer`となりました。`Bearer`は6文字で、スペースデリミタも削除する必要があるため、最初の7文字をスライスしています。AWS Cognitoのようにタイプを提供しない認証サービスもありますが、そのようなサービスでは、`jwt`関数がトークン値だけを受け取れるように、スライス数が変更されるか、完全に削除されます。

NGINX のコア構成内で、NJS モジュールをロードします。`http`ブロック内でJavaScriptをインポートして使用します：

```
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;
    ...
}
```

このNGINX構成は、`njs`モジュールを動的にロードし、以前に定義したJavaScriptファイルをインポートします。NGINXディレクティブは、JavaScript関数の戻り値にNGINX変数を設定するために使用されます。

この変数を用いて、JavaScriptのロジックを証明します。JavaScriptによって設定された変数を返すサーバーを定義します：

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;
    server_name _;
    location / {
        return 200 "$jwt_payload_subject $jwt_payload_issuer";
    }
}
```

この構成は、クライアントがAuthorizationヘッダーを介して提供したsubjectとissuerの値を返すサーバーを生成します。これらの値は、定義されたJavaScriptコードによってデコードされます。

コードが動作することを検証するために、指定されたJWTでサーバーに要求を送ります。以下はJWTのJSON形式であり、このコードが動作することを検証するために使用できます：

```
{
  "iss": "nginx",
  "sub": "alice",
  "foo": 123,
  "bar": "qq",
  "zyx": false
}
```

与えられたJWTでサーバーに要求を行い、JavaScriptのコードが実行され、正しい値が返されることを確認します：

```
$ curl 'http://localhost/' -H \
"Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1\
NiIsImV4cCI6MTU4NDcyMzA4NX0.eyJpc3MiOiJuZ2lueCIsInN1YiI6Im\
FsaWNlIiwiZm9vIjoxMjMsImJhciI6InFxIiwie\
nl4IjpmYWxzZX0.Kftl23Rvv9dIso1RuZ8uHaJ83BkKmMtTwch09rJtwgk"

alice nginx
```

解説

NGINXは、要求と応答の処理中に標準的なJavaScriptの機能を公開するモジュールを提供しています。このモジュールにより、ビジネスロジックをプロキシ層に埋め込むことができます。JavaScriptはその使用量の多さから選ばれました。

njsモジュールは、NGINXへの要求とNGINXからの応答の間にロジックを注入する機能を提供しています。このセクションで示すように、JWTをデコードすることで、プロキシを通過するときに要求を検証および操作できるようになります。また、njsモジュールは、JavaScriptロジックを介して応答データをストリーミングすることで、アップストリームサービスからのレスポンスを操作できます。njsはさらに、「関連項目」セクションで詳しく説明されているように、ストリームサービスをアプリケーション層として認識させることができます。

関連項目

njsスクリプト言語ドキュメンテーション

NGINX Plus njsモジュールのインストール

5.4 共通プログラミング言語でNGINXを拡張

問題

一般的なプログラミング言語を使用して独自の拡張を実行する必要があります。

解決法

Cプログラミング言語を用いて独自のNGINXモジュールを最初から作成する準備をする前に、まず、他のプログラミング言語モジュールの1つがユースケースに適合するかどうかを評価します。Cプログラミング言語は非常に強力で高性能です。ただし、必要なカスタマイズを実現するモジュールとして利用できる言語は他にも多数あります。NGINXは、NGINX JavaScript（njs）を発表しました。これは、モジュールを有効にするだけでJavaScriptの機能をNGINX構成に統合できます。Lua/Perlモジュールも利用できます。これらの言語モジュールを使用して、コードを含むファイルをインポートするか、構成内でコードのブロックを直接定義します。これらの言語モジュールを使用して、コードを含むファイルをインポートするか、構成内でコードのブロックを直接定義します。

Luaを使用するには、Luaモジュールと次のNGINX構成をインストールして、Luaスクリプトをインラインで定義します：

```
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")
            }
        }
    }
}
```

Luaモジュールはngxという名前のモジュールで定義されるオブジェクトを介して独自のNGINX APIを提供します。njsのrequestオブジェクトと同様に、ngxオブジェクトには応答を記述し、応答を操作する属性とメソッドがあります。

Perlモジュールをインストールした場合のこの例では、Perlの実行環境からNGINX変数を設定するためPerlを使用します。

```
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
            }
        }
    }
}
```

前述の例は、これらの言語モジュールが単に応答を返すだけでなく、より多くの機能を提供することを示します。perl_setディレクティブはPerlスクリプトから返されたデータにNGINX変数を設定します。前述の例は、これらの言語モジュールが単に応答を返すだけでなく、より多くの機能を提供することを示します。

解説

NGINXの拡張性によって実現される機能は無限大です。NGINXは、Cモジュールを介してカスタムコードで拡張できます。Cモジュールは、ソースから構築する際にNGINXにコンパイルしたり、構成内で動的に読み込みしたりできます。JavaScript (njs)、Lua、Perlの機能と構文を公開する既存のモジュールはすでに利用可能です。)、Lua、Perl の機能と構文を公開する既存のモジュールはすでに利用可能です。多くの場合、独自のNGINX機能が必要にならない限り、これらの既存のモジュールで十分です。これらのモジュール用に構築された多くのスクリプトが、オープンソースコミュニティにすでにあります。

ここに記載の解決法は、NGINXで利用可能なLuaおよびPerlスクリプト言語の基本的な使用法を示しました。応答、変数の設定、サブリクエストの作成、複雑な書き換えの定義などを希望する場合、これらのNGINXモジュールがその機能を提供します。

関連項目

NGINX Plus Luaモジュールドキュメンテーション

NGINX Plus Perlモジュールドキュメンテーション

NGINX Luaモジュールドキュメンテーション

NGINX Perlモジュールドキュメンテーション

5.5 Ansibleを使ったインストール

問題

Ansibleを使用してNGINXをインストール、構成し、NGINX構成をコードとして管理し、その他のAnsible構成に準拠する必要があります。

解決法

Ansible GalaxyからAnsible NGINXコレクションをインストールします：

```
ansible-galaxy collection install nginxinc.nginx_core
```

NGINXコレクションとnginxロールを使用してNGINXをインストールするプレイブックを作成します：

```
---
- hosts: all
  collections:
    - nginxinc.nginx_core
  tasks:
    - name: Install NGINX
      include_role:
        name: nginx
```

ビルドインのnginx_configロールを使用するタスクをプレイブックに追加し、ユースケースに合わせてデフォルトテンプレートに変数のオーバーライドを提供します：

```
- name: Configure NGINX
  ansible.builtin.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
    deployment_location: /usr/share/nginx/html/index.html
    web_server_name: Ansible NGINX collection
```

解説

Ansibleは、Pythonをベースにした、広く使用されているパワフルな構成管理ツールです。 タスクの構成はYAMLで行われ、ファイルのテンプレートにはJinja2テンプレート言語を使用します。Ansibleは、サブスクリプションモデルのAnsible Automation Platformというサーバーを提供します。しかし、通常Ansibleはローカルマシンからの実行、サーバーを直接構築するためクライアントへの実行、スタンドアロンでの実行で利用されます。Ansibleはサーバーに対しSecure Shell（SSH）で接続し、構成ファイルの内容を実行します。他の構成管理ツールと同様に、パブリックにロールを共有する大規模なコミュニティがあります。Ansibleのコミュニティは Ansible Galaxyと呼ばれます。プレイブックで利用できる非常に洗練されたロールを見つけることができます。

この解決法では、F5, Inc.が保守するパブリックロールのコレクションを使用し、NGINXをインストールして、サンプル構成を作成しました。この例で使用されている構成は、NGINXのデモHTMLファイルをテンプレート化し、*/usr/share/nginx/html/index.html*に配置します。また、NGINX構成ファイルは、`localhost:80`でリッスンするサーバーブロックを生成するようテンプレート化されており、単一のロケーションブロックがデモファイルを提供するようにも構成されています。提供されているNGINX構成テンプレートは非常に包括的ですが、`nginx_config`ロールを使用すると、事前構築および保守されているAnsible構成を活用しながら、独自のテンプレートを提供して完全に制御できます。また、F5,Incは、WAFとDosのためのNGINX App Protectモジュールのインストールと構成に使用できるNGINX App Protectロールを保守しています。

関連項目

NGINX提供のAnsibleコレクション

Ansibleドキュメンテーション

5.6 Chefを使ったインストール

問題

Chefを使用してNGINXをインストール、構成し、NGINX構成をコードとして管理し、その他のChef構成に準拠する必要があります。

解決法

Sous ChefsによってNGINXクックブックをChefスーパーマーケットからインストールします：

```
$ knife supermarket install nginx
```

このクックブックはリソースベースです。つまり、独自のクックブックで使うためのChefリソースを提供しています。自身のNGINXユースケースに合わせてクックブックを作ってください。このクックブックには、スーパーマーケットからインストールされたnginxクックブックが依存関係として含まれます。依存関係が存在すると、提供されたリソースを使用できます。NGINXをインストールするためのレシピを作成します：

```
nginx_install 'nginx' do
  source 'repo'
end
```

sourceがrepoに設定されている場合、NGINXはF5, Inc.が保守するリポジトリからインストールされ、最新版が提供されます。

レシピ内でnginx_configリソースを使用して、コアNGINX構成設定を上書きします：

```
nginx_config 'nginx' do
  default_site_enabled true
  keepalive_timeout 65
  worker_processes 'auto'
  action :create
  notifies :reload, 'nginx_service[nginx]', :delayed
end
```

レシピ内でnginx_siteリソースを使用して、NGINXサーバーブロックを構成します

```
nginx_site 'test_site' do

  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
```

解説

Chefは、Rubyで作成された構成管理ツールです。クライアント/サーバー構成で実行することも、単独の構成でも実行できます。Chefには、パブリックのクックブックや、スーパーマーケットと呼ばれる非常に大きなコミュニティがあります。スーパーマーケットによるパブリッククックブックはKnifeと呼ばれるコマンドラインユーティリティを使ってインストール/保守可能です。Chefは非常に多機能であり、ここに示したレシピはほんの小さなサンプルです。

スーパーマーケットのパブリックNGINXクックブックは柔軟性が高く、パッケージマネージャーまたはソースからNGINXを簡単にインストールするオプションや多数の異なるモジュールをコンパイルしてインストールする機能、そして基本構成をテンプレート化する機能を提供します。このセクションでは、Sous Chefsが保守するリポジトリからNGINXをインストールし、基本的な例としてHTMLファイルをホストするサーバーブロックを構成しました。`nginx_site`リソースに独自のテンプレートを提供することで、NGINX構成をさらに制御できます。

関連項目

Chefドキュメンテーション

Chef Supermarket for NGINX

5.7 Consulテンプレートによる構成の自動化

問題

Consulを使用して環境の変化に対応するために、NGINX構成を自動化する必要があります。

解決法

consul-templateデーモンとテンプレートファイルを使用して、選択したNGINX構成ファイルをテンプレート化します：

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

この例は、アップストリーム構成ブロックをテンプレート化するConsulテンプレートファイルです。このテンプレートは、app.backendとして識別されるConsul内のノードをループします。Consulの各ノードにテンプレートはそのノードのIPアドレスを使用してサーバーディレクティブを生成します。

consul-templateデーモンはコマンドラインを介して実行され、構成ファイルが変更されてテンプレート化されるたびにNGINXをリロードするために使用できます：

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

このコマンドはconsul-templateデーモンにconsul.example.internalのConsulクラスタに接続し、現在の作業ディレクトリにある*upstream.template*という名前のファイルを使用してファイルをテンプレート化し、生成されたコンテンツを*/etc/nginx/conf.d/upstream.conf*に出力し、その後、テンプレート化されたファイルに変更が生じるたびにNGINXをリロードするよう指示します。-templateフラグは、テンプレートファイルの文字列、出力ロケーション、テンプレート化プロセスの実行後に実行するコマンドを受け取ります。これらの3つの変数は、コロンで区切られています。実行中のコマンドにスペースが含まれている場合は、必ず二重引用符 で囲んでください。-consulフラグは、どのConsulクラスタに接続するかをデーモンに指示します。

解説

Consulは、強力なサービス検出ツールおよび構成ストアです。Consulは、ノードとキー値のペアに関する情報をディレクトリのような構造に格納し、RESTful APIインタラクションを可能にします。 Consulは、各クライアントにDNSインターフェースも提供し、クラスタに接続されているノードのドメイン名をルックアップできるようにします。Consulクラスタを利用する別の プロジェクトはconsul-templateデーモンです。このツールは、Consulノード、サービス、またはキーバリューのペアの変更に応じてファイルをテンプレート化します。この機能のためNGINXの自動化を考慮する場合、Consulはパワフルな選択肢となっています。consul-templateでテンプレートへの変更が行われた後にコマンドを実行するようにデーモンに指示することもできます。これにより、NGINX構成をリロードし、NGINX構成を環境とともに有効にすることができます。Consulとconsul-templateを使用することで、NGINX構成は環境と同じく動的になることができます。インフラストラクチャ、構成、およびアプリケーションの情

報は一元的に保存され、`consul-template`は必要に応じてイベントベースの方法でサブスクライブおよび再テンプレート化できます。このテクノロジーにより、NGINXはサーバー、サービス、アプリケーションバージョンなどの追加と削除に応じて動的に再構成できます。

関連項目

NGINX Plusのサービス検出統合によるロードバランシング

NGINXおよびConsulテンプレートによるロードバランシング

Consulホームページ

Consulテンプレートによるサービス構成

CconsulテンプレートGitHub

第6章
認証

6.0 はじめに

GINXはクライアントを認証できます。NGINXを使用したクライアントの認証は作業を減らし、アプリケーションサーバーに認証されていない要求が到達するのを停止する能力を提供します。NGINXオープンソースで利用できるモジュールには基本的な認証と認証サブリクエストが含まれます。JSON Web Token（JWT）を検証するためのNGINX Plus専用モジュールにより、認証標準のOpenID Connectを使用するサードパーティの認証プロバイダーとの統合が可能になります。JSON Web Token（JWT）を検証するためのNGINX Plus専用モジュールを使用することで、OpenID Connect認証標準を使用するサードパーティの認証プロバイダーとの統合が可能になります。本章では、NGINXを認証と組み合わせて使用してリソースを保護するさまざまな方法について説明します。

6.1 HTTP Basic認証

問題

HTTP Basic認証を介して、アプリケーションまたはコンテンツを保護する必要があります。

解決法

次の形式でファイルを生成し、パスワードは、許可されている形式のいずれかで暗号化またはハッシュするようにします：

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

ユーザー名は最初のフィールド、パスワードは2番目のフィールド、区切り文字はコロンです。オプションの3番目のフィールドがあり、ここは各ユーザーへのコメントとして使用できます。NGINXは、複数の異なる形式のパスワードを理解できます。そのうちの1つは、C言語の関数

crypt()で暗号化されたパスワードです。

この関数openssl passwdコマンドによってコマンドラインに公開されます。代わりにopensslを使用すると、以下のコマンドで暗号化されたパスワード文字列を作成できます：

```
$ openssl passwd MyPassword1234
```

出力は、NGINXがパスワードファイル内で使用できる文字列になります。

auth_basicおよびauth_basic_user_fileディレクティブをNGINX構成の中で使用してBasic認証を有効にします：

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

http、server、またはlocationコンテキストでauth_basicディレクティブを使用することができます。auth_basicディレクティブは文字列パラメータを取ります。これは、認証されていないユーザーが到達したときにBasic認証ポップアップウィンドウに表示されます。auth_basic_user_fileはユーザーファイルへのパスを指定します。

構成をテストするために、curlと-uまたは--userフラグを使用して要求にAuthorizationヘッダーを構築します：

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

解説

ベーシック認証パスワードは、セキュリティのレベルを変更し、いくつかの方法と異なる形式で生成できます。Apacheからのhtpasswdコマンドもパスワードを生成できます。opensslおよびhtpasswdコマンドのどちらもNGINXが理解できるapr1アルゴリズムのパスワードを生成できます。パスワードはLightweight Directory Access Protocol（LDAP）やDovecotが使用するSalted SHA-1形式も使用可能です。NGINXは多数の形式とハッシュアルゴリズムをサポートしています。しかし、ブルートフォース攻撃によって簡単に侵害される可能性があるため、それらの多くは安全でないと見なされます。

Basic認証を使用して、NGINXホスト全体、特定の仮想サーバー、または特定のロケーションブロックのコンテキストを保護できます。Basic認証は、Webアプリケーションのユーザー認証に取って代わるものではありませんが、個人情報を安全に保つのに役立ちます。あまり知られていませんが、ベーシック認証はサーバーが応答ヘッダーWWW-Authenticateを含む401の認証資格不足のHTTPコードを返すことによって行われます。このヘッダーにはBasic realm="*your string*"の値があります。この応答により、ブラウザはユーザー名とパスワードの入力を求めます。ユーザー名とパスワードは連結され、コロンで区切られ、base64でエンコードされてから、Authorizationという名前の要求ヘッダーで送信されます。Authorization要求ヘッダーは、Basicおよびuser:passwordでエンコードされた文字列を指定し、提供されたauth_basic_user_fileに対して照合します。ユーザー名とパスワードの文字列はbase64でエンコードされているだけであり、ユーザー認証情報はインターネットを介して平文で送信される

ので、Basic認証ではHTTPSを使用することをお勧めします。

6.2 認証サブリクエスト

問題

要求の認証を希望するサードパーティの認証システムがあります。

解決法

`http_auth_request_module`を使用して認証サービスに要求に応える前にIDを検証するよう要求します。

```
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;
}
```

`auth_request`ディレクティブはURIパラメータを取ります。このパラメータはローカルの内部ロケーションでなければなりません。`auth_request_set`ディレクティブは認証サブリクエストから変数を設定することを許可します。.

解説

`http_auth_request_module`はNGINXサーバーが処理するすべての要求で認証を有効にします。モジュールはサブリクエストを使用して、リクエストの続行が許可されているかどうかを判断します。サブリクエストとは、NGINXが要求を別の内部ロケーションに渡し、要求を宛先にルーティングする前にその応答を観測する時です。`/auth location`ブロックは本文とヘッダーを含め元の要求を認証サーバーに渡します。サブリクエストのHTTPステータスコードは、アクセスが許可されるかどうかを決定するものです。サブリクエストがHTTP200ステータスコードで返される場合、認証は成功し、要求は実行されます。サブリクエストがHTTP 401または403を返す時、同じものが元の要求に返されます。

認証サーバーが要求本文を求めない場合、例示のとおり、`proxy_pass_request_body`ディレクティブで要求本文を削除できます。この行動により要求のサイズと時間が削減されます。応答本文は破棄されるため、`Content-Length`ヘッダーは空の文字列に設定する必要があります。認証サービスが要求で指定しているURIを知る必要がある場合は、認証サービスがチェックおよび検証するカスタムヘッダーにその値を記す必要があります。応答ヘッダーやその他の情報な

ど、サブリクエストから認証サービスまで保持したいものがある場合は、auth_request_setディレクティブを使用して、応答データから新しい変数を作成できます。

6.3 NGINX PlusでのJWTの照合

問題

要求がNGINX Plusに処理される前にJWTを照合する必要があります。

解決法

NGINX PlusのHTTP JWT認証モジュールを使用して、トークンの署名を照合し、JWTクレームとヘッダーをNGINX変数として埋め込みます。

```
location /api/ {
  auth_jwt "api";
  auth_jwt_key_file conf/keys.json;
}
```

この構成はこのロケーションのJWTの照合を有効にします。auth_jwtディレクティブには、認証レルムとして使用される文字列が渡されます。auth_jwtは、JWTの情報を保持する変数をオプションのトークンパラメータとして指定します。デフォルトでは、Authenticationヘッダーが JWTで使用されます。auth_jwtディレクティブは継承された構成から必要なJWT認証の効果を相殺するためにも使用されます。認証をオフにするには、auth_jwtディレクティブを使用し、パラメータなしで渡します。継承された認証要件を相殺するには、offキーワードだけをauth_jwtディレクティブに渡します。auth_jwt_key_fileは単一のパラメータを取ります。このパラメータは、標準のJSON Web Key（JWK）形式のキーファイルへのパスです。

NGINX Plus R29では、レート制限と組み合わせてJWT使用することで未署名のJWTによるサービス拒否攻撃から保護する、nginx_http_internal_redirectという名前のモジュールが追加されました。このモジュールは、JWTチェックの前にレート制限のチェックを行うことで、不正アクターからの保護を強化します。以下に、internal_redirectディレクティブとその使用例を示します。

```
limit_req_zone $jwt_claim_sub zone=jwt_sub:10m rate=1r/s;

server {
  location / {
    auth_jwt "realm";
    auth_jwt_key_file key.jwk;

    internal_redirect @rate_limited;
  }

  location @rate_limited
{
```

```
        internal;

        limit_req zone=jwt_sub burst=10;
        proxy_pass http://backend;
    }
}
```

解説

NGINXPlusは、トークン全体が暗号化されるJSON Web Encryptionタイプではなく、Jason Web Signatureタイプのトークンを照合できます。NGINX Plusは、HS256、RS256、およびES256アルゴリズムで署名された署名を照合できます。NGINX Plusにトークンを照合させることで、認証サービスへのサブリクエストを行うために必要な時間とリソースを節約できます。NGINX Plusは、JWTヘッダーとペイロードを解読し、標準のヘッダーとクレームを埋め込み変数にキャプチャして使用できるようにします。`auth_jwt`ディレクティブは、`http`、`server`、`location`、`limit_except`コンテキストで使用できます。このセクションでは、未署名のJWTによるサービス拒否攻撃から保護するNGINX Plusの`internal_redirect`モジュールの使い方についても説明しました。このモジュールの詳細については、「関連項目」セクションを参照してください。

関連項目

JSON Web SignatureのRFC標準ドキュメンテーション

JSON Webアルゴリズムの RFC標準ドキュメンテーション

JSON Webトークンの RFC標準ドキュメンテーション

NGINX Plus JWT認証

ブログ（英語）：“Authenticating API Clients with JWT and NGINX Plus”

NGINX internal_redirectモジュール

6.4 JSON Web Keyの作成

問題

NGINX Plusが使用するJSON Web Key（JWK）が必要です。

解決法

NGINX Plusは、RFC標準で指定されているJWK形式を利用します。この標準では、JWKファイル内のキーオブジェクトの配列が許可されています。

以下は、キーファイルがどういった見かけかを示す例です：

```
{"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"
    }
  ]
}
```

例示のJWKファイルは、RFC標準に記載されている3つの初期タイプのキーを表します。これらのキーの形式もRFC標準の一部です。kty属性がキーの種類です。このファイルは3つの主要なタイプを示します。対称鍵（oct）、楕円曲線鍵（EC）およびRSAです。kid属性はキー IDです。このファイルは、対称鍵（oct）、楕円曲線鍵（EC）およびRSAの3つの主要なタイプを示します。これらのキーの他の属性は、そのタイプのキーの標準で指定されています。詳細については、これらの標準のRFCドキュメントを参照してください。

解説

さまざまな言語で利用できるライブラリが多数存在し、JWKを生成するために使用できます。JWKを定期的に作成およびローテーションするための、JWKの中心的な権限である主要なサービスを作成することをお勧めします。セキュリティを強化するために、JWKをSSL/TLS証明書と同様に安全にすることをお勧めします。キーファイルを適切なユーザーやグループ権限で安全に保護してください。それらをホストのメモリに保持することがベストプラクティスです。ramfsのようなメモリ内ファイルシステムを作成してこれを実行できます。一定の間隔でキーをローテーションさせることも重要です。公開鍵と秘密鍵を作成し、APIを介してアプリケーションとNGINXに提供する鍵サービスを作成することも選択できます。

関連項目

JSON Web KeyのRFC標準ドキュメンテーション

6.5 NGINX Plusでの既存のOpenID Connect SSOを介したユーザー認証

問題

NGINX PlusをOpenID Connect（OIDC）IDプロバイダーと統合する必要があります。

解決法

この解決法は、多くの設定項目とNGINX JavaScript（njs）の少しのコードで構成されています。IDプロバイダー（IdP）はOpenID Connect 1.0をサポートする必要があります。NGINX Plusは、認可コードフローにおいて、OIDCのリライングパーティとして機能します。

F5は、NGINX PlusとのOIDC統合のリファレンス実装として、設定とコードを含むパブリックGitHubリポジトリを保持しています。「関連項目」セクションのリポジトリへのリンクには、お使いのIdPとのリファレンス実装のセットアップ方法に関する最新の手順が記載されています。

解説

この解決法は、読者が最新の解決法を確実に入手できるように、リファレンス実装にリンクしているだけです。提供されているリファレンスは、OpenID Connect 1.0の認可コードフローのリライングパーティとしてNGINX Plusを構成します。この構成で、保護されたリソースに対する認証されていない要求がNGINX Plusに対して行われると、NGINX Plusは最初に要求をIdPにリダイレクトします。IdPは、クライアントに独自のログインフローを実行させ、認証コードを使用してクライアントをNGINX Plusに戻します。次に、NGINX PlusはIdPと直接通信して、認証コードをIDトークンのセットと交換します。

これらのトークンはJWTを使用して照合され、NGINX Plusのキーバリューストアに保存されます。キーバリューストアを使用することにより、トークンは、高可用性（HA）構成のすべてのNGINX Plusノードで使用できるようになります。このプロセス中に、NGINX Plusは、キーバリューストアでトークンを検索するためのキーとして使用されるクライアントのセッションcookieを生成します。次に、クライアントには、最初に要求されたリソースへのcookieを使用したリダイレクトが提供されます。後続の要求は、cookieを使用してNGINX Plusのキーバリューストアで IDトークンをルックアップして照合されます。

この機能により、CA SingleSignOn（以前のSiteMinder）、ForgeRockOpenAM、Keycloak、Okta、OneLogin、PingIdentityなどのほとんどの主要なIDプロバイダーとの統合が可能になります。標準としてのOIDCは、認証に非常に関連しています。前述のIDプロバイダーは、実現可能な統合の一部にすぎません。

関連項目

ブログ（英語）：“Authenticating Users to Existing Applications with OpenID Connect and NGINX Plus”

OpenID Connect

NGINX OpenID Connect GitHub

OIDC Connectのアクセストークンサポート

6.6 NGINX PlusでのJSON Webトークン（JWT）の照合

問題

NGINX PlusでJSON Webトークンを照合する必要があります。

解決法

NGINX Plusに付属のJWTモジュールを使用して、ロケーションまたはサーバーを保護し、照合するトークンとして`$cookie_auth_token`を使用するように`auth_jwt`ディレクティブに指示します。

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

この構成は、JWT照合を使用して/private/URIパスを保護するようにNGINX Plusに指示します。Google OAuth 2.0 OpenID Connectは、デフォルトのベアラートークンではなく`auth_token`を使用します。そのため、NGINXPlusのデフォルトの場所ではなく、このcookieの中のトークンを探すようにNGINXに指示する必要があります。`auth_jwt_key_file`のロケーションは任意のパスに設定されます。これはレシピ6.7で説明するステップです。

解説

この構成は、Google OAuth 2.0 OpenID ConnectJWTをNGINX Plusで照合する方法を示します。NGINX Plus JWT認証モジュール（HTTP用）は、JSON Web Signature仕様のためのRFCに準拠するすべてのJWTを照合できるため、JWTを利用するすべてのSSO機関をNGINX Plusレイヤーですぐに照合できます。OpenID 1.0プロトコルは、IDを追加するOAuth 2.0認証プロトコルの上にあるレイヤーであり、JWTを使用して要求を送信するユーザーのIDを証明できるようにします。トークンの署名により、NGINX Plusは、トークンが署名されてから変更されていないことを検証できます。この方法で、Googleは非同期署名方式を使用しており、プライベートJWKをシークレットに保ちながら、パブリックJWKを配布することを可能にしています。

関連項目

“Authenticating API Clients with JWT and NGINX Plus”

6.7 NGINX PlusでJSON Webキーセットを自動的に取得してキャッシュする

問題

NGINX PlusがプロバイダーにJSON Webキーセット（JWKS）を自動的に要求し、キャッシュするようにする必要があります。

解決法

キャッシュゾーンとauth_jwt_key_requestディレクティブを利用して、キーを自動的に最新の状態に保ちます：

```
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;
  }
}
```

この例では、`auth_jwt_key_request`ディレクティブは、内部サブリクエストからJWKSを取得するようにNGINX Plusに指示します。サブリクエストは*/jwks_uri*に向けられ、IDプロバイダーに要求をプロキシします。オーバーヘッドを制限するために、要求はデフォルトで10分間キャッシュされます。

解説

NGINX Plus R17では、`auth_jwt_key_request`ディレクティブが導入されました。この機能により、NGINX Plusサーバーは、要求が行われた際にJWKを動的に更新できます。サブリクエストメソッドがJWKを取得するために使用されるため、ディレクティブが指す場所はNGINX Plusサーバーに対してローカルである必要があります。この例では、NGINX Plusの内部リクエストのみが処理されるように、サブリクエストの場所がロックダウンされています。

キャッシュは、JWKの取得要求が必要な頻度でのみ行われるように使用されました。このため、IDプロバイダーに過負荷がかかることはありません。

`auth_jwt_key_request`ディレクティブは`http`、`server`、`location`、`limit_except`コンテキスト

で有効です。

関連項目

ブログ（英語）：“Authenticating API Clients with JWT and NGINX Plus”

ブログ（英語）：NGINX Plus “Faster JWT Validation with JSON Web Key Set Caching”

6.8 SAML認証のサービスプロバイダーとしてのNGINX Plusの設定

問題

NGINX PlusをSAML IDプロバイダー（IdP）と統合してリソースを保護する必要があります。

解決法

この解決法では、NGINX Plusキーバリューストアが使用されるので、NGINX Plusにのみ適用できます。また解決法では、NGINX JavaScript（njs）モジュールも使用されます。njsのインストールから始めます。

APTパッケージマネージャーでNGINX Plusをインストール：

```
$ apt install nginx-plus-module-njs
```

YUMパッケージマネージャーでNGINX Plusをインストール：

```
$ yum install nginx-plus-module-njs
```

これで、njsが動的モジュールとしてインストールされます。*nginx.conf*構成に以下を追加して、モジュールをロードするようにNGINX Plusに指示する必要があります。

```
load_module modules/ngx_http_js_module.so;
```

この機能を有効にするには、SAML JavaScriptおよびNGINX Plus構成ファイルをダウンロードして解凍し、適切な場所に移します。

```
$ wget https://github.com/nginxinc/nginx-saml/archive/refs/heads/main.zip \
    -O nginx-saml-main.zip
$ unzip nginx-saml-main.zip
$ mv nginx-saml-main/* /etc/nginx/conf.d/
```

このNGINX Plus SAMLの解決法ではまだ、標準的なSAML XML構成ファイルは解析されません。前のステップでダウンロードした構成ファイルを更新して、システムと統合する必要があります。以下は、各ファイルの説明と、更新が必要な内容を示します。

saml_sp_configuration.conf

1つ以上のサービスプロバイダー（SP）とIdPのプライマリ構成を`map{}`ブロックに含めます。マッピング機能を使用して、$host変数に基づき複数のSPやIdPを設定できます。

- SPの設定に合わせて、`$saml_sp_`で始まる変数をレンダリングするmapブロックのすべてを変更します。
- IdPの設定に合わせて、`$saml_idp_`で始まる変数をレンダリングするmapブロックのすべてを変更します。
- `$saml_log_out_redirect`変数をレンダリングするmapブロックで定義されるURIを変更して、`/logout`の場所を要求した後に表示される保護されていないリソースを指定します。
- NGINX Plusが他のプロキシやロードバランサーの後ろに配置されている場合、`$redirect_base`変数と`$proto`変数のマッピングを変更して、元のプロトコルとポート番号の取得方法を定義します。

frontend.conf

以下のリバースプロキシの構成例では、*saml_sp.server_conf*ファイルをインポートすることで、SAMLによる解決法を使用してリソースを保護します。

- `include conf.d/saml_sp.server_conf;`をサーバー構成内に複製します。
- 要求が不正な場合にSAML SPフローを開始する`error_page`ディレクティブを使用します。保護する`location`ブロックに以下を配置します。

```
error_page 401 = @do_samlsp_flow;
if ($saml_access_granted != "1") {
return 401;
}
```

- アップストリーム要求に、認証されたユーザー名を含むヘッダーを設定します。

```
proxy_set_header username: $saml_name_id;
```

- オプションで、`error_log`ディレクティブに渡される`level`パラメータを更新して、ログレベルを更新します。

saml_sp.server_conf

IdP応答を処理するためのNGINX構成：

- このファイルでの変更は通常必要ありません。
- 最適化のために、`client_body_buffer_size`ディレクティブをIdP応答（POSTボディ）の最大サイズに合わせて変更します。

saml_sp.js

SAML認証を実行するJavaScriptコード：

- 変更は必要ありません。

解説

この解決法では、NGINX JavaScriptモジュールとNGINX Plusキーバリューストアモジュールを組み合わせることで、NGINXがSAMLサービスプロバイダー（SP）としてリソースをシングルサインオンで保護できるようにします。F5のNGINXチームは、JavaScriptコードをGitHubの公開リポジトリという形式で提供しています。これは、構成にインストールして有効にする必要があります。エンドポイントとキーの構成、およびその他のSAML構成は、一連の`map`ブロックによってJavaScriptコードに提供されます。これにより、処理される要求のホスト名に基づいて、複数のSPまたはIdPを設定できます。

SPとIdPが構成されると、*saml_sp.server_conf*をサーバー構成に追加して、`error_page`ディレクティブにより、NGINX Plusにユーザーのセッションが保存されていない場合にIdPとのSPフローを開始できるようになります。SPフローは、ユーザーをIdPに送り、既存のセッションまたはログインを使用します。これに成功すると、ユーザーはNGINX Plusによって検証され、キーバリューストアに格納されるSAML応答とともにNGINX Plusにリダイレクトされます。SAML応答のキーは、以降の要求で使用されるcookie形式でクライアントに返されます。最後に、クライアントが、最初に要求されたリソースに戻されます。

この解決法では、SAMLシングルログアウト（SLO）もサポートしています。これにより、ユーザーは1回の操作ですべてのSPとIdPからログアウトできます。この操作は、SP主導でも、IdP主導でも可能です。SP主導のログアウトは、NGINX PlusがIdPにLogoutRequestメッセージを送信することで始まります。その後、IdPがユーザーのセッションを終了し、LogoutResponseメッセージをNGINX Plusに返します。次に、NGINX Plusがこのユーザーセッションのキーバリューストアに格納されている値を削除します。

IdP主導のログアウトの場合、IdPが、SPとしてNGINX Plusに連絡し、登録されているLogout URLにLogoutRequestを送信することでログアウト処理を開始します。この処理が実行されると、NGINX Plusは、ユーザーセッションに関連付けられたキーバリューストアから値を削除し、LogoutResponseメッセージをIdPに返します。

SLO機能は、IdPによりサポートされていない場合、またはNGINX Plus SPがその使用を許可しない場合、無効にすることができます。SLOを無効にするには、`$saml_idp_slo_url`の変数マップ構成に空文字列を設定します。

関連項目

NGINX PlusのSAML SSOサポート

第7章
セキュリティコントロール

7.0 はじめに

セキュリティはレイヤーで行われ、セキュリティモデルを真に強化するには、セキュリティモデルに複数のレイヤーが必要です。

本章では、NGINXを使用してWebアプリケーションを保護するためのさまざまな方法について説明します。これらのセキュリティ対策の多くを組み合わせて使用すると、セキュリティを強化できます。

お気付きになるかもしれませんが、本章では、NGINXをWebアプリケーションファイアウォール（WAF）に変える機能である、ModSecurity 3.0 NGINXモジュールについては説明していません。WAF機能の詳細については、クイックガイド:*ModSecurity 3.0 and NGINX*をご覧ください。

NGINX ModSecurity WAF for NGINX Plusは、2024年3月31日付けで生産終了（EoL）に移行しますのでご注意ください。詳しくは、こちらのブログ「F5 NGINX ModSecurity WAF Is Transitioning to End-of-Life」をご覧ください。

7.1 IPアドレスに基づくアクセス

問題

クライアントの送信元に基づきアクセスをコントロールする必要があります。

解決法

HTTPまたはストリームアクセスモジュールを使用して、保護対象のリソースへのアクセスをコントロールします。

```
location /admin/ {
  deny 10.0.0.1;
  allow 10.0.0.0/20;
```

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

例示のロケーションブロックは、10.0.0.1を除く10.0.0.0/20の任意のIPv4アドレスからのアクセスを許可し、2001:0db8::/32サブネットのIPv6アドレスからのアクセスを許可し、他のアドレスから発信された要求に対して403を返します。 allowおよびdenyディレクティブは、http、サーバー、ロケーションコンテキスト、また、TCP/UDPのストリームおよびサーバーコンテキスト内で有効です。

リモートアドレスに一致するものが見つかるまで、ルールが順番にチェックされます。

解説

インターネット上の貴重なリソースとサービスの保護は、複数のレイヤーで行う必要があります。NGINXの機能は、そうしたレイヤーのうちの1つになる機能を提供します。denyディレクティブは、特定のコンテキストへのアクセスをブロックし、allowディレクティブは、ブロックされたアドレスのサブセットへのアクセスを許可するために使用できます。IPアドレス、IPv4またはIPv6、クラスレスドメイン間ルーティング（CIDR）ブロック範囲、キーワードall、およびUnixソケットを使用できます。通常、リソースを保護する場合、内部IPアドレスのブロックを許可し、すべてからのアクセスを拒否する場合があります。

7.2 クロスオリジンソース共有の許可

問題

別のドメインからリソースを提供しており、ブラウザがこれらのリソースを利用できるようにするには、クロスオリジンリソース共有（CORS）を許可する必要があります。

解決法

CORSを有効にするには、requestメソッドに基づいてヘッダーを変更します。

```
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;
    }
  }
}
```

この例ではたくさんのことが行われていますがmapを使用してGETメソッドとPOSTメソッドをグループ化することで要約されています。OPTIONSリクエストメソッドは、このサーバーのCORSルールに関して*preflight*リクエストをクライアントに返します。CORSに基づき、OPTIONS、GET、POSTメソッドは許可されています。Access-Control-Allow-Originヘッダーを設定すると、このサーバーから提供されるコンテンツを、このヘッダーに一致するオリジンのページでも使用できるようになります。プリフライトリクエストは、クライアントに1,728,000秒、つまり20日間キャッシュできます。

解説

JavaScriptなどのリソースは、リクエストしているリソースがそれ自体以外のドメインのものである場合にCORSを作成します。リクエストがクロスオリジンと見なされる場合、ブラウザはCORSルールに従う必要があります。特に使用を許可するヘッダーがない場合、ブラウザーはリソースを使用しません。リソースをその他のサブドメインで使用できるようにするには、CORSヘッダーを設定する必要があります。これはadd_headerディレクティブで実行できます。標準コンテンツタイプのGET、HEAD、POSTリクエストのいずれかの場合で、リクエストに特別なヘッダーがない場合には、ブラウザはリクエストを行い、発信元のみをチェックします。他のリクエストメソッドの場合、ブラウザはプリフライトリクエストを実行して、対象のリソースについてサーバーが求める条件を確認する必要があります。 これらのヘッダーを適切に設定しないと、そのリソースを利用しようとした場合にブラウザでエラーが発生します。

7.3 クライアント側の暗号化

問題

NGINXサーバーとクライアント間のトラフィックを暗号化する必要があります。

解決法

`ngx_http_ssl_module`や`ngx_stream_ssl_module`などのSSLモジュールを使用してトラフィックを暗号化します：

```
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;
  }
}
```

この構成では、SSL/TLS、8443で暗号化されたポートでリッスンするようにサーバーを設定します。`ssl_certificate`ディレクティブは、証明書と、クライアントに提供されるオプションのチェーンを定義します。ディレクティブ`ssl_certificate`は、証明書と、クライアントに提供されるオプションのチェーンを定義します。`ssl_certificate_key`ディレクティブは、リクエストを復号化し、レスポンスを暗号化するためにNGINXが使用するキーを定義します。多くのSSL/TLSネゴシエーション構成は、デフォルト設定で適用されています。

解説

安全なトランスポート層は、転送中の情報を暗号化する最も一般的な方法です。この記事の執筆時点では、SSLプロトコルよりもTLSプロトコルが好まれています。これは、SSLのバージョン1から3が安全ではないと見なされるようになったためです。NGINXバージョン1.23.4以降では、デフォルトのSSLプロトコルは、TLSv1、TLSv1.1、TLSv1.2およびTLSv1.3（OpenSSLライブラリでサポートされる場合）です。プロトコル名は異なる場合がありますが、TLSは依然として安全なソケット層を確立します。NGINXを使用すると、サービスでユーザーとクライアント間の情報を保護できます。これにより、クライアントとビジネスが保護されます。

CA署名付き証明書を使用する場合は、証明書を認証局チェーンと連結する必要があります。証明書とチェーンを連結する場合、証明書は連結されたチェーンファイル上になければなりません。認証局がチェーンの中間証明書として複数のファイルを提供している場合、それらを階層化する順序があります。その順序については、証明書プロバイダーのドキュメントを参照してください。

関連項目

Mozillaセキュリティ/サーバー側TLSページ

Mozilla SSL 構成ジェネレータ

SSL 構成テストを SSL サーバテストで実行

7.4 クライアント側の暗号化（応用）

問題

高度なクライアント/サーバー暗号化構成が必要です。

解決法

NGINXのhttpおよびstreamSSLモジュールにより、受け入れられたSSL/TLSハンドシェイクを完全にコントロールできます。証明書とキーは、ファイルパスまたは変数の値を通じてNGINXに指定できます。NGINXは、構成ごとに、受け入れられたプロトコル、暗号、キータイプのリストをクライアントに提示します。クライアントとNGINXサーバー間の最高水準がネゴシエートされます。NGINXは、クライアント/サーバーのSSL/TLSネゴシエーションの結果を一定期間キャッシュできます。

以下では、クライアント/サーバーネゴシエーションの利用可能な複雑さを説明する目的で、一度に多くのオプションを意図的に示しています。

```
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;
  }
}
```

サーバーは、SSLプロトコルバージョンTLSv1.2およびTLSv1.3を受け入れます。受け入れられる暗号はHIGHに設定されます。これは暗号強度の高いサイファーを示すマクロです。aNULLとMD5については、!の表記により明示的な拒否を示します。

2セットの証明書とキーのペアが使用されます。NGINXディレクティブに渡される値は、NGINX証明書キー値を提供するさまざまな方法を示します。変数はファイルへのパスとして解

釈されます。data:プレフィックスが付いている場合は直接値として解釈されます。クライアントに下位互換性を提供するために、複数の証明書キー形式が提供される場合があります。ネゴシエーション結果によりクライアントとサーバーとの最良の接続が確立されます。

SSL/TLSキーが直接値変数として公開されている場合、構成によってログに記録されたり公開されたりする可能性があります。キー値を変数として公開する場合は、厳密な変更とアクセスコントロールがあることを確認してください。

SSLセッションキャッシュとタイムアウトにより、NGINXワーカープロセスはセッションパラメータを一定時間キャッシュして保存できます。NGINXワーカープロセスは、このキャッシュを単一のインスタンス化内のプロセスとして相互に共有しますが、マシン間では共有しません。すべてのタイプのユースケースにおいて、パフォーマンスやセキュリティに役立つ他の多くのセッションキャッシュオプションがあります。

セッションキャッシュオプションを相互に組み合わせて使用できます。しかし、デフォルトなしでセッションキャッシュを指定すると、デフォルトのビルドインセッションキャッシュがオフになります。

解説

この高度な例では、NGINXは、TLSv1.2または1.3のSSL/TLSオプション、高く評価されている暗号アルゴリズム、RSAまたは楕円曲線暗号（ECC）形式のキーを使用する機能をクライアントに提供します。

クライアントが実行できる最も強力なプロトコル、暗号、キー形式は、ネゴシエーションの結果です。構成は、10 MBの使用可能なメモリー割り当てで10分間ネゴシエーションをキャッシュするようにNGINXに指示します。

テストでは、ECC証明書は同等の強度のRSA証明書よりも高速であることがわかりました。キーサイズが小さいため、より多くのSSL/TLS接続を提供でき、ハンドシェイクが高速になります。NGINXを使用すると、複数の証明書とキーを構成して、クライアントブラウザに最適な証明書を提供できます。これにより、新しいテクノロジーを利用しながら、古いクライアントにサービスを提供できます。

この例では、NGINXはクライアントとのトラフィックを暗号化しています。ただし、アップストリームサーバーへの接続も暗号化される場合があります。NGINXとアップストリームサーバー間のネゴシエーションは、レシピ7.5に示されています。

関連項目

Mozillaセキュリティ/サーバー側TLSページ

Mozilla SSL 構成ジェネレータ

SSLラボのSSLサーバーテスト

7.5 アップストリーム暗号化

問題

NGINXとアップストリームサービス間のトラフィックを暗号化し、コンプライアンス規制のための特定のネゴシエーションルールを設定する必要があります。そうしなければ、アップストリームサービスがセキュリティで保護されないネットワーク通信になります。

解決法

HTTPプロキシモジュールのSSLディレクティブを使用して、SSLルールを指定します：

```
location / {
  proxy_pass https://upstream.example.com;
  proxy_ssl_verify on;
  proxy_ssl_verify_depth 2;
  proxy_ssl_protocols TLSv1.3;
}
```

これらのプロキシディレクティブは、NGINXが従う、特定のSSLルールを設定します。構成されたディレクティブにより、NGINXはアップストリームサービスの証明書とチェーンが最大2つの証明書の範囲まで有効であることを検証するようになります。`proxy_ssl_protocols`ディレクティブはNGINXがTLSバージョン1.3のみ使用するよう指定します。

デフォルトでは、NGINXはアップストリーム証明書を検証せず、すべてのTLSバージョンを受け入れます。

解説

HTTPプロキシモジュールの設定ディレクティブは多数あり、アップストリームトラフィックを暗号化する必要がある場合は、少なくとも検証をオンにする必要があります。proxy_passディレクティブに渡される値のプロトコルを変更するだけで、HTTPSを介してプロキシできます。ただし、これはアップストリーム証明書を検証しません。`proxy_ssl_certificate`や`proxy_ssl_certificate_key`のようなその他のディレクティブは、一層にセキュリティのためにアップストリーム暗号化をロックダウンすることを可能にします。また、`proxy_ssl_crl`または証明書失効リストを指定することもできます。このリストには、無効と見なされなくなった証明書が表示されます。これらのSSL `proxy`ディレクティブは、独自のネットワーク内またはパブリックインターネット全体でシステムの通信チャネルを強化するのに役立ちます。

7.6 ロケーションの保護

問題

シークレットを使ってlocationブロックを保護する必要があります。

解決法

セキュアリンクモジュールとsecure_link_secretディレクティブを使用して、リソースへのアクセスをセキュアリンクを持つユーザーに制限します：

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

  rewrite ^ /secured/$secure_link;
}

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

この構成により、内部および公開のロケーションブロックが作成されます。公開ロケーションブロック/resourcesはリクエストURIに、secure_link_secretディレクティブに提供されたシークレットで検証できるmd5ハッシュ文字列が含まれていない限り、403 Forbiddenが返されます。URIのハッシュが検証されない限り、$secure_link変数は空の文字列です。

解説

シークレットを使用してリソースを保護することは、ファイルを確実に保護するための優れた方法です。シークレットはURIとの組み合わせで使用されます。次に、この文字列はmd5ハッシュされ、その md5ハッシュの16進ダイジェストがURIで使用されます。ハッシュはリンク内に配置され、NGINXによって評価されます。NGINXは、ハッシュの後のURIにあるため、要求されているファイルへのパスを認識しています。NGINXは、secure_link_secretディレクティブを介して提供されるため、あなたのシークレットも認識します。NGINXはmd5ハッシュをすばやく検証し、URIを$secure_link変数に格納できます。ハッシュを検証できない場合、変数は空の文字列に設定されます。secure_link_secretに渡される引数は静的な文字列でなければならない点に注意してください。変数にすることはできません。

7.7 秘密を用いたセキュアリンクの生成

問題

シークレットを使用してアプリケーションからセキュアリンクを生成する必要があります。

解決法

NGINXのセキュアリンクモジュールはmd5ハッシュ文字列の16進ダイジェストを受け入れます。この文字列は、URIパスとシークレットを連結したものです。最後のセクションに基づいて、*/var/www/secured/ index.html*にファイルがあると仮定してレシピ7.6、前の構成例で機能するセキュアなリンクを作成します。 md5ハッシュの16進ダイジェストを生成するには、Unixの

openssl コマンドを使用できます：

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

ここでは、保護しているURI、*index.html*を、シークレットと連結した状態で示します：シークレットはmySecretです。この文字列はopensslコマンドに渡され、md5の16進ダイジェストが出力されます。

以下は、Python標準ライブラリに含まれているhashlibライブラリを使用してPythonで構築されている同じハッシュダイジェストの例です：

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

このハッシュダイジェストができたので、URLで使用できます。ここでの例は*www.example.com*が/resources ロケーションを介してファイル*/var/www/secured/index.html*をリクエストします。完全なURLは次のとおりです：

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

解説

ダイジェストの生成は、さまざまな方法で、多くの言語で実行できます。留意点：URIパスはシークレットの前に配置され、文字列にはキャリッジリターンがなく、md5ハッシュの16進ダイジェストを使用してください。

7.8 有効期限のあるロケーションの保護

問題

将来のいつかの時点でに期限切れになるクライアントに固有のリンクを使用して、ロケーションを保護する必要があります。

解決法

セキュアリンクモジュールに含まれている他のディレクティブを利用して、有効期限を設定し、セキュアリンクで変数を使用します：

```
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; }
}
```

secure_linkディレクティブはカンマ区切りの2つのパラメータを取ります。最初のパラメータはmd5ハッシュを持つ変数です。この例では、md5のHTTP引数を使用しています。 2番目のパラメータは、リンクが期限切れになる時間をUnixエポック時間形式で保持する変数です。secure_link_md5ディレクティブはmd5ハッシュの生成に使用される文字列の形式を宣言する1つのパラメータを取ります。他の構成同様に、ハッシュが検証しない場合には、$secure_link変数は空の文字列に設定されます。しかし、この使用事例の場合、ハッシュが一致して、時間が失効している場合$secure_link変数は0に設定されます。

解説

リンクを保護するこの使用法は、レシピ 7.6に示されているsecure_link_secretよりも柔軟性があり、見た目も簡潔です。これらのディレクティブを使用すると、ハッシュ文字列でNGINXで使用できる変数を数にかかわらず使用できます。ハッシュ文字列でユーザー固有の変数を使用すると、ユーザーはセキュリティで保護されたリソースへのリンクを交換できないため、セキュリティが強化されます。$remote_addrまたは$http_x_forwarded_forなどの変数、またはアプリケーションによって生成されたセッションcookieヘッダーを使用することが推奨されます。secure_linkの引数は、任意の変数から取得でき、最適な名前を付けることができます。条件：アクセスはありますか？設定された時間内にアクセスしますか？アクセスがない場合：禁止。アクセスがあるものの、遅れた：失効。HTTP410「ページが削除されました」は失効したリンクに一番適しています。この状態は永久だと認識されているためです。

7.9 有効期限のあるリンクの生成

問題

有効期限のあるリンクを生成する必要があります。

解決法

Unixエポック形式で有効期限のタイムスタンプを生成します。Unixシステムでは、次に示される方法で日付を使用してテストできます：

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

次に、secure_link_md5ディレクティブで構成された文字列と一致するように、ハッシュ文字列を連結する必要があります。このケースでは使用される文字列は1924905600/resources/index.html127.0.0.1 mySecretです。md5ハッシュは、単なる16進ダイジェストとは少し異なります。これは、base64でエンコードされたバイナリ形式のmd5ハッシュで、プラス記号（+）がハイフン（-）に変換され、スラッシュ（/）がアンダースコア（_）に変換され、等号（=）が削除されています。以下は、Unixシステムでの例です：

```
$ echo -n '1924905600/resources/index.html127.0.0.1 mySecret' \
  | openssl md5 -binary \
  | openssl base64 \
  | tr +/ -_ \
  | tr -d =
sqysOw5kMvQBL3j9ODCyoQ
```

これでハッシュができたので、有効期限とともに引数として使用できます：

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

以下は、有効期限の相対時間を利用した、Pythonで生成から1時間で有効期限が切れるようにリンクを設定する場合のより実用的な例です。本書執筆の時点では、この例はPython標準ライブラリを利用してPython 2.7および3.xで動作します：

```
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)
```

解説

このパターンを使用すると、URLで使用できる特別な形式でセキュアリンクを生成できます。シークレットは、クライアントに送信されることのない変数を使用してセキュリティを提供します。ロケーションを保護するために必要な他の変数をいくつでも使用できます。`md5`ハッシュとbase64エンコーディングは一般的で軽量なため、ほぼすべての言語で利用できます。

7.10 HTTPSリダイレクト

問題

暗号化されていない要求をHTTPSにリダイレクトする必要があります。

解決法

書き換えを使用して、すべてのHTTPトラフィックをHTTPSに送信します。

```
server {
  listen 80 default_server;
  listen [::]:80 default_server;
  server_name _;
  return 301 https://$host$request_uri;
}
  server {
  listen 443 ssl;
  listen [::]:443 ssl;
  ssl_certificate /etc/nginx/ssl/example.crt;
  ssl_certificate_key /etc/nginx/ssl/example.key;
  ...
}
```

この構成は、IPv4とIPv6の両方、および任意のホスト名のデフォルトサーバーとしてポート`80`でリッスンします。`return`ステートメントはHTTPSサーバーに301永続的リダイレクトを同じホストと要求URIで返します。これは、SSL/TLSに構成されたサーバーブロックにより処理されます。

解説

必要に応じて、常にHTTPSにリダイレクトすることが重要です。すべての要求をリダイレクトする必要はなく、クライアントとサーバー間で機密情報が渡される要求のみをリダイレクトする必要がある場合があります。その場合、`return`ステートメントを*/login*などの特定のロケーションにのみ配置することをお勧めします。

7.11 NGINXより手前でSSL/TLSが終端されている場合のHTTPSへのリダイレクト

問題

HTTPSにリダイレクトする必要がありますが、NGINXの手前のレイヤーでSSL/TLSを終端しています。

解決法

共通の`X-Forwarded-Proto`ヘッダーを使用して、リダイレクトする必要があるかどうかを判断します：

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

この構成は、HTTPSリダイレクトと非常によく似ています。しかし、この構成では、ヘッダー`X-Forwarded-Proto`がHTTPと同じ場合にのみリダイレクトします。

解説

この構成は、追加費用なしでSSL/TLSをオフロードするAmazon Web Services Elastic Load Balancing（AWS ELB）などのレイヤーで機能します。これは、HTTPトラフィックが保護されていることを確認するための手軽な方法です。

7.12 HTTPストリクトトランスポートセキュリティ

問題

ブラウザにHTTPで要求を送信しないよう指示する必要があります。

解決法

`Strict-Transport-Security`ヘッダーを設定して、HTTP Strict Transport Security（HSTS）拡張機能を使用します。

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

この構成では、`Strict-Transport-Security`ヘッダーが最長1年に設定されます。これにより、このドメインへのHTTP要求が試行された場合に常に内部リダイレクトを実行するようにブラウザに指示されるため、すべての要求はHTTPSを介して行われます。

解説

一部のアプリケーションでは、中間者攻撃にトラップされた1つのHTTP要求が問題になる可能性があります。機密情報を含むフォーム投稿がHTTP経由で送信された場合、NGINXからのHTTPSリダイレクトによって対応することはできません。損害が生じます。この選択式のセキュリティ拡張機能は、HTTP要求を行わないようにブラウザに通知するため、要求が暗号化されずに送信されることはありません。

関連項目

RFC Standard Documentation of HTTP StrictのRFC標準ドキュメンテーション

OWASP HTTP Strict Transport Securityチートシート

7.13 国に基づくアクセス制限

問題

契約上の理由から、またはアプリケーションの要件のために、特定の国からのアクセスを制限する必要があります。

解決法

該当するディストリビューションのNGINX公式リポジトリからNGINX GeoIPモジュールをインストールします。

NGINX Plus：

```
$ apt install nginx-plus-module-geoip
```

NGINX Open Source：

```
$ apt install nginx-module-geoip
```

ブロックまたは許可する国コードを変数にマップします：

```
load_module
  "/etc/nginx/modules/ngx_http_geoip_module.so";

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

このマッピングは新しい変数 `$country_access` を1または0に設定します。クライアントのIPアドレスが、USまたはカナダからの場合、変数は0に設定されます。その他の国の場合には、

すべて1に設定されます。

では、serverブロック内でifステートメントを使用してUSまたはカナダ以外の国からのアドレスによるアクセスを拒否してみましょう：

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

このifステートメントは$country_access変数が1に設定されている場合にはTrueと評価されます。Trueの場合、サーバーは403未認証を返します。それ以外の場合にはサーバーは通常通り機能します。そのため、このifブロックは、USまたはカナダ以外の国からのアドレスによるアクセスを拒否するためだけに存在します。

解説

これは、一部の国からのアクセスのみを許可する方法に関する簡単な例です。この例はニーズに合わせて応用できます。同じ方法をGeoIPモジュールで利用できる任意の埋め込み変数に基づき許可またはブロックする場合に利用できます。

関連項目

NGINX Geo-IPモジュールドキュメンテーション

7.14 任意の数のセキュリティ方法を満たす

問題

クローズドサイトにセキュリティを追加するために、複数の方法を提供する必要があります。

解決法

satisfyディレクティブを使用して、NGINXに使用されるセキュリティメソッドのいずれか、または全てを満たしたいことを指示します。

```
location / {
  satisfy any;

  allow 192.168.1.0/24;
  deny all;
  auth_basic "closed site";
  auth_basic_user_file conf/htpasswd;
}
```

この構成は、location /を要求するユーザーがセキュリティメソッドの1つを満たす必要があることをNGINXに通知します。要求は*192.168.1.0/24*CIDRブロックから発信されるか、*conf/*

*htpasswd*ファイルに含まれるユーザー名とパスワードを提供できなければなりません。`satisfy`ディレクティブは、2つのオプションのいずれかを取ります：`any`または`all`。

解説

`satisfy`ディレクティブは、Webアプリケーション認証のための複数の方法を提供する優れた方法です。`satisfy`ディレクティブに`any`を指定することにより、ユーザーはセキュリティ上の課題の1つを満たす必要があります。`satisfy`ディレクティブに`all`を指定することによりユーザーはセキュリティ上の課題のすべてを満たす必要があります。このディレクティブは、レシピ 7.1で詳細が説明された`http_access_module`、inレシピ6.1で詳細が説明された`http_auth_basic_module` レシピ 6.2で詳細が説明された`http_auth_request_module`、レシピ 6.3で詳細が説明された`http_auth_jwt_module`と一緒に使用できます。セキュリティは、複数のレイヤーで行われる場合にのみ真に安全です。`satisfy`ディレクティブは、重大なセキュリティルールを必要とするロケーションとサーバーに対してこれを適用するのに役立ちます。

7.15 NGINX Plus動的アプリケーションイヤーによるDDoS攻撃の軽減

問題

動的な分散型サービス拒否（DDoS）軽減ソリューションが必要です。

解決法

NGINX App Protect DoSモジュールまたはNGINX Plus機能を使用して、クラスタアウェアのレート制限と自動ブロックリストを作成します：

```
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/9/http/keyvals/sinbin break;
              # Set the URI of the
              # "sin bin" key-val
        proxy_method POST;
        proxy_set_body '{"$remote_addr":"1"}';
        proxy_pass http://127.0.0.1:80;
    }

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

解説

この解決法は、同期されたキーバリューストアにより同期されたレート制限を使用します。DDoS攻撃に動的に応答し、その影響を軽減します。または、NGINX App Protectを使用する場合、DoSモジュールによりこの機能が提供されます。`limit_req_zone`および`keyval_zone`ディレクティブに提供されるsyncパラメータは、共有メモリーゾーンを NGINX Plusクラスタ内の他のマシンとアクティブ-アクティブで同期します。この例では、どのNGINX Plusノードが要求を受信するかに関係なく、1秒あたり100を超える要求を送信するクライアントを特定します。クライアントがレート制限を超えると、NGINX Plus APIを呼び出すことにより、そのIPアドレスが「sin bin」キーバリューストアに追加されます。「sin bin」は、クラスタ全体で同期されます「sin bin」のクライアントからのそれ以上の要求は、どのNGINX Plusノードがそれらを受信するかに関わらず、非常に低い帯域幅制限の対象となります。帯域幅を制限することは、要求を完全に拒否するよりも望ましい方法です。DDoS軽減が適用されていることをクライアントに明確に通知しないためです。10分後、クライアントは自動でsin binから削除されます。

関連項目

F5, Inc. NGINX App Protect DoS

7.16 NGINX Plus App Protect WAFモジュールのインストールと構成

問題

NGINX App Protect WAFモジュールをインストールして構成する必要があります。

解決法

NGINX App Protect WAF管理ガイドのお使いのプラットフォームの項目に従ってください。別のリポジトリからNGINX App Protect WAF署名をインストールする部分をスキップしないように注意してください。

メインコンテキストで`load_module`ディレクティブを使用してNGINX App Protect WAFモジュールがNGINX Plusによって動的に読み込まれ、`app_protect_*`ディレクティブを使用することによって有効になっていることを確認します：

```
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
}
```

この例では、`app_protect_enable`ディレクティブは現在のコンテキストにおいてモジュールを設定することで機能を有効にできます。このディレクティブ、およびこれ以下のすべては、`http`コンテキスト内、およびHTTPを使用する`server`コンテキストと`location`コンテキスト内で有効です。`app_protect_policy_file`ディレクティブは、次に定義するNGINX App Protect WAFポリシーファイルを指定します。定義されていない場合は、デフォルトのポリシーが使用されます。セキュリティログが次に構成されますが、これにはリモートログサーバーが必要です。この例では、ローカルのSyslogサーバを宛先として指定します。`app_protect_security_log`ディレクティブは2つのパラメータを取ります。1つめはログ設定を定義するJSONファイルで、2つめはログストリームの宛先です。ログ設定ファイルは、このセクションの後半で示します。

NGINX App Protect WAFポリシーファイルを構築し、*/etc/nginx/AppProtect-TransparentPolicy.json*という名前を付けます：

```
{
  "policy": {
    "name": "transparent_policy",
    "template": { "name": "POLICY_TEMPLATE_NGINX_BASE" },
    "applicationLanguage": "utf-8",
    "enforcementMode": "transparent"
  }
}
```

このポリシーファイルは、テンプレートを使用してデフォルトのNGINX App Protect WAFポリシーを構成し、ポリシー名を`transparent_policy`に設定し、`enforce mentMode`を`transparent`に設定します。これは、NGINX Plusがログに記録するが、通信を拒否しないことを意味します。.トランスペアレントモードは、新しいポリシーを有効にする前にテストする場合に最適です。

`enforcementMode`から`blocking`に変更して、ブロッキングを有効にします。このポリシーファイルは*/etc/nginx/AppProtectBlockingPolicy.json*という名前にできます。ファイルを切り替えるには、NGINX Plus構成の`app_protect_policy_file`ディレクティブを更新します：

```
{
  "policy": {
    "name": "blocking_policy",
    "template": { "name": "POLICY_TEMPLATE_NGINX_BASE" },
    "applicationLanguage": "utf-8",
    "enforcementMode": "blocking"
  }
}
```

NGINX App Protect WAFの保護機能の一部を有効にするには、一部のVIOLATIONを有効にします：

```
{
  "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
        }
      ]
```

```
      }
    }
  }
```

上記の例は、ポリシーに2つのVIOLATIONを追加する方法を示します。.VIOL_PARAMETER_VALUE_ METACHARはブロックに設定されておらず、警告だけするよう設定されている一方でVIOL_JSON_FORMATはブロックして警告を出すよう設定されている点にご注意ください。この機能により、blockingに設定されている場合、デフォルトのenforcementModeオーバーライドできますenforcementModeが透明に設定されている場合、デフォルトの強制設定が優先されます。

/etc/nginx/log-default.json という名前のNGINX Plusログファイルを設定します。

```
{
  "filter":{
    "request_type":"all"
  },
  "content":{
     "format":"default",
     "max_request_size":"any",
     "max_message_size":"5k"
  }
}
```

このファイルは、app_protect_security_logディレクティブによってNGINX Plusの設定で定義されており、NGINX App Protect WAFのログを記録するために必要となります。

解説

この解決策は、NGINX PlusとNGINX App Protect WAFモジュールの構成の基礎を示します。NGINX App Protect WAFモジュールは、Webアプリケーションファイアウォール（WAF）の定義全体を有効にします。これらの定義は、F5, Inc.の高度なF5アプリケーションセキュリティ機能から派生しています。

この包括的なWAF攻撃シグネチャのセットは、広範囲にわたってフィールドテストが行われ、実証されています。これをNGINX Plusインストールに追加することで、最高のF5アプリケーションセキュリティとNGINXプラットフォームの俊敏性を組み合わせが実現します。

モジュールがインストールされて有効になると、ほとんどの構成はポリシーファイルで行われます。このセクションのポリシーファイルは、アクティブブロッキング、パッシブモニタリング、透明モードを有効にする方法を示し、VIOLATIONによるこの機能のオーバーライドについて説明しました。 V違反は、提供される保護の1つのタイプにすぎません。その他の保護には、HTTPコンプライアンス、回避技術、攻撃シグネチャ、サーバーテクノロジー、データガードなどが含まれますNGINX App Protect WAFログを取得するには、NGINX Plusログ形式を使用して、ログをリモートリスニングサービス、ファイル、または/dev/stderrに送信する必要があります。NGINX Controller ADCを使用している場合は、NGINX Controllers App Securityコンポーネントを介してNGINX App Protect WAF機能を有効にし、Webインターフェースを介してWAFメトリクスを視覚化できます。

関連項目

NGINX App Protect WAF 管理ガイド

NGINX App Protect WAF 構成ガイド

NGINX App Protect DoS 導入ガイド

第8章
HTTP/2およびHTTP/3（QUIC）

8.0 はじめに

HTTP/2およびHTTP/3は、HTTPプロトコルの主要な改訂です。これらのアップデートはどちらも、ヘッドオブラインブロッキングとして知られるパフォーマンス問題に対処するために設計されています。この問題は、先にキューにあるネットワークパケットが完了するまで後続のパケットを送信できないというものです。HTTP/2では、1つのTCPコネクション上でリクエストとレスポンスの完全な多重化を可能にすることで、アプリケーション層の最初のTCPハンドシェイクにおけるヘッドオブラインブロッキングの問題が緩和されています。しかし、TCPプロトコル自体では、その損失パケット回復メカニズムにより、パケットを順番に送信することと、損失したパケットをブロックして再送してから処理を進めることが必要であるため、依然としてこの問題は発生します。HTTP/3では、TCPはユーザーデータグラムプロトコル（UDP）に基づくQUIC（「クイック」と発音します）プロトコルに置き換えられました。UDPは、コネクションハンドシェイク、送信順序、または損失パケット回復の実装が必要ないので、QUICはこれらのトランスポートの課題をより効率的な方法で解決できます。

また、HTTP/2では、HTTPヘッダーフィールドの圧縮とリクエストの優先順位付けのサポートが導入されています。本章では、NGINXでHTTP/2およびHTTP/3を有効にするための基本的な構成と、Googleのオープンソースリモートプロシージャコール（gRPC）の構成について詳しく説明します。

8.1 HTTP/2の有効化

問題

HTTP/2を利用したいと考えています。

解決法

NGINXサーバーでHTTP/2をオンにします：

```
server {
  listen 443 ssl default_server;
  http2 on;

  ssl_certificateserver.crt;
  ssl_certificate_key server.key;
  # ...
}
```

解説

HTTP/2をオンにするには、`http2`ディレクティブをオンに設定するだけです。ただし、プロトコルでは接続をSSL/TLSでラップする必要はありませんが、HTTP/2クライアントの一部の実装では、暗号化された接続でHTTP/2のみがサポートされます。 もう1つの注意点は、HTTP/2仕様が多数のTLSv1.2暗号スイートをブロックしているため、ハンドシェイクに失敗することです。NGINXがデフォルトで使用する暗号は、ブロックリストに含まれていません。TLSのアプリケーション層プロトコルネゴシエーションにより、アプリケーション層は、追加のラウンドトリップを回避するために、安全な接続を介してどのプロトコルを使用するかをネゴシエートできます。セットアップが正しいことをテストするには、サイトがHTTP/2を使用していることを示す、ChromeおよびFirefoxブラウザ用のプラグインをインストールします。または、コマンドラインで`nghttp`ユーティリティを使用します。

関連項目

HTTP/2 RFC Cipher Suite Black List

Chrome HTTP/2およびSPDY Indicator Plug-in

Firefox HTTP Version Indicator Add-on

8.2 HTTP/3の有効化

問題

QUICプロトコル経由でHTTP/3を利用する必要があります。

解決法

NGINXサーバーでQUICを有効にして、`Alt-Svc`ヘッダーを返すことでプロトコルが利用可能であることをクライアントに通知します。

```
server {
  # for better compatibility we recommend
  # using the same port number for QUIC and TCP
  listen 443 quic reuseport; # QUIC
  listen 443 ssl; # TCP
```

```
    ssl_certificate certs/example.com.crt;
    ssl_certificate_key certs/example.com.key;
    ssl_protocols TLSv1.3;

    location / {
      # advertise that QUIC is available on the configured port
      add_header Alt-Svc 'h3=":$server_port"; ma=86400';
      # ...
    }
  }
```

解説

HTTP/3をオンにするには、quicパラメータをlistenディレクティブに追加する必要があります。同じポートでTCP接続をリッスンするようにNGINXに指示する別のlistenディレクティブを設定することもできます。TLSv1.3はQUICプロトコルで必要なため、TLSv1.3のみをネゴシエートするようにssl_protocolディレクティブを設定します。

要求されたHTTPサービスがHTTP/3経由で利用可能であることをクライアントに知らせるために、Alt-Svcと呼ばれるHTTPヘッダーが返されます。クライアントは、最初にサービスを要求するときに、HTTP/1.1を使用して、このヘッダーを含むレスポンスを受け取ります。この値は、利用できるサービス（この場合HTTP/3のh3）、およびサービスを見つけるポート番号をクライアントに知らせます。私たちの例では、同じポートでTCPとUDPをリッスンしているので、ハードコード化された値を減らすために$server_port変数を使用できます。返されるポート番号はクライアントが接続するポート番号でなければなりません。これは、裏でポートマッピングを行っている場合はNGINXがリッスンしているポート番号とは異なる可能性があるので注意してください。そのサービスが利用可能になるまでの時間も定義されます。これにより、クライアントは指定された時間（この例では86400秒（24時間））の間、このサービスに対するUDPリクエストを継続します。

NGINXオープンソースバージョン1.25.2およびNGINX Plus R30では、--withhttp_v3_moduleがデフォルトモジュールになりました。これらのバージョン以前では、HTTP/3を使用するには、代替のSSLライブラリと--withhttp_v3_module構成フラグを使用してNGINXをソースからコンパイルする必要がありました。本書執筆の時点では、OpenSSLはQUICのTLSインターフェースをサポートしていません。F5, Inc.のNGINXチームは、OpenSSL互換レイヤーを開発し、quictls、BoringSSL、LibreSSLのようなサードパーティのTLSライブラリをビルドして出荷する必要がなくなりました。

関連項目

「NGInX Plus R30の発表」

ブログ（英語）：“A Primer on QUIC Networking and Encryption in NGINX”

「QUIC+HTTP/3対応のNGINXプレビュー実装版バイナリパッケージのご紹介」

NGINX HTTP V3モジュールドキュメンテーション

8.3 gRPC

問題

gRPCメソッド呼び出しを終端、検査、ルーティングまたはロードバランシングする必要があります。

解決法

NGINXを使用してgRPC接続をプロキシします：

```
server {
  listen 80;

  http2 on;
  location / {
    grpc_pass grpc://backend.local:50051;
  }
}
```

この構成では、NGINXは暗号化されていないHTTP/2トラフィックをポート`80`でリッスンし、そのトラフィックをポート`50051`で`backend.local`という名前のマシンにプロキシします。`grpc_pass` ディレクティブは、通信をgRPC呼び出しとして扱うようにNGINXに指示します。バックエンドサーバーの場所の前にある`grpc://`は必要ありませんが、バックエンド通信が暗号化されていないことを直接示しています。

クライアントとNGINXの間でTLS暗号化を利用し、アプリケーションサーバーに呼び出しを渡す前にその暗号化を終了するには、最初のセクションで行ったように、SSLとHTTP/2をオンにします。

```
server {
  listen 443 ssl default_server;

  http2 on;
  ssl_certificateserver.crt;
  ssl_certificate_key server.key;
  location / {
    grpc_pass grpc://backend.local:50051;
  }
}
```

この構成は、NGINXでTLSを終端し、暗号化されていないHTTP/2を介してgRPC通信をアプリケーションに渡します。

アプリケーションサーバーへのgRPC通信を暗号化するようにNGINXを構成し、エンドツーエンドの暗号化トラフィックを提供するには、サーバー情報の前に`grpcs://`を指定するように`grpc_pass`ディレクティブを変更するだけです（安全な通信を示すsの追加を確認します）。

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

NGINXを使用して、パッケージ、サービス、メソッドを含むgRPCURIに基づいてさまざまなバックエンドサービスに呼び出しをルーティングすることもできます。これを行うには、locationディレクティブを使用します：

```
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;
}
```

この構成例では、locationディレクティブを使用して、2つの別々のgRPCサービス間で着信HTTP/2トラフィックをルーティングし、静的コンテンツを提供するロケーションも使用します。mypackage.service1サービスのメソッド呼び出しは、ホスト名またはIPとオプションのポートを含む可能性のある変数grpc_service1の値に送信されます。mypackage.service2の呼び出しは変数grpc_service2の値に向けられます。.location /は他のHTTPリクエストをキャッチし、静的コンテンツを提供します。これは、NGINXが同一のHTTP/2エンドポイントの下で、gRPCと非gRPCをに対応し、それぞれに応じてルーティングする方法を示します。

gRPC呼び出しのロードバランシングは非gRPC HTTPトラフィックの場合と類似しています：

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

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

upstreamブロックは、他のHTTPトラフィックの場合とまったく同じようにgRPCで機能します。唯一の違いは、upstreamがgrpc_passによって参照されることです。

解説

NGINXは、gRPC呼び出しの暗号化を受信、プロキシ、ロードバランシング、ルーティング、終端することができます。gRPCモジュールは、NGINXがgRPC呼び出しヘッダーを設定、変更、削除を行い、要求のタイムアウトを設定し、アップストリームSSL/TLS仕様を設定できるようにします。gRPCはHTTP/2プロトコルを介して通信するため、同じエンドポイントでgRPCと

非gRPCのWebトラフィックを受け入れるようにNGINXを構成できます。

第9章
高度なメディアストリーミング

9.0 はじめに

本章では、NGINXを使用したMPEG-4（MP4）またはFlash Video（FLV）形式のストリーミングメディアについて説明します。NGINXは、コンテンツを大量のクライアントに配信およびストリーミングするために広く使用されています。NGINXは、本章で説明する業界標準のフォーマットとストリーミングテクノロジーをサポートしています。NGINX Plusを使用すると、HTTPライブストリーミング（HLS）モジュールを使用して自動でコンテンツをフラグメント化する機能と、既にフラグメント化されたメディアのHTTPダイナミックストリーミング（HDS）を配信する機能が提供されます。NGINXは帯域幅制限を標準で許可し、NGINX Plusの高度な機能はビットレート制限を提供し、サーバーのリソースを予約してほとんどのユーザーに接続できるように、コンテンツを最も効率的な方法で配信できるようにします。

9.1 MP4とFLV

問題

MP4またはFLVで作成されたデジタルメディアをストリーミングする必要があります。

解決法

.mp4 または *.flv* ビデオを提供するHTTPロケーションブロックを指定します。NGINXは、プログレッシブダウンロードまたはHTTP疑似ストリーミングを使用してメディアをストリーミングし、シークをサポートします。

```
http {
  server {
    # ...

    location /videos/ {
```

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

例のロケーションブロックはNGINXに、ビデオディレクトリ内のファイルはMP4形式タイプであり、プログレッシブダウンロードサポートを使用してストリーミングできると告げます。2番目のロケーションブロックは、*.flv*で終わるファイルはすべてFLV形式であり、HTTP疑似ストリーミングサポートを使用してストリーミングできることをNGINXに指示します。

解説

NGINXでのビデオまたはオーディオファイルのストリーミングは、1つのディレクティブと同じくらい簡単です。プログレッシブダウンロードを使用すると、クライアントはファイルのダウンロードが完了する前にメディアの再生を開始できます。NGINXは、両方の形式でメディアのダウンロードされていない部分のシークをサポートします。

9.2 NGINX PlusでのHLSストリーミング

問題

MP4ファイルにパッケージ化されたH.264/AACエンコードコンテンツのHTTP Live Streaming（HLS）をサポートする必要があります。

解決法

リアルタイムのセグメンテーション、パケット化、多重化を備えたNGINX PlusのHLSモジュールを利用し、HLS引数の転送など、フラグメンテーションバッファリングなどをコントロールします：

```
  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;
  }
```

このlocationブロックはNGINXに*/var/www/video*ディレクトリからHLSメディアをストリー

ムして、4秒セグメントにメディアをフラグメンテーションするよう指示します。HLSバッファ数は10、10 MBに設定されています。初期のMP4バッファサイズは1 MBに設定され、最大5 MBです。

解説

NGINX Plusで利用可能なHLSモジュールは、MP4メディアファイルを自動でトランスマルチプレックス（多重化）する機能を提供します。メディアをフラグメント化およびバッファリングする方法を制御できる多くのディレクティブがあります。ロケーションブロックは、HLSハンドラーを使用してメディアをHLSストリームとして提供するように構成する必要があります。HLSフラグメンテーションは秒数で設定され、NGINXに時間の長さでメディアをフラグメント化するように指示します。バッファされるデータの量は、バッファの数とサイズを指定する`hls_buffers`ディレクティブで設定されます。クライアントは、`hls_mp4_buffer_size`で指定された一定量のバッファが発生した後、メディアの再生開始が許可されます。ただし、ビデオに関するメタデータが初期バッファサイズを超える可能性があるため、より大きなバッファが必要になる場合があります。この量は、`hls_mp4_max_buffer_size`によって制限されます。これらのバッファリング変数により、NGINXはエンドユーザーエクスペリエンスを最適化できます。これらのディレクティブに適切な値を選択するには、ターゲットオーディエンスとメディアを知る必要があります。たとえば、メディアの大部分が大きなビデオファイルであり、ターゲットオーディエンスの帯域幅が広い場合は、最大バッファサイズを大きくし、フラグメントを長くすることを選択できます。これにより、コンテンツに関するメタデータを最初にエラーなしでダウンロードし、ユーザーがより大きなフラグメントを受信できるようになります。

9.3 NGINX PlusでのHDSストリーミング

問題

すでにフラグメント化され、メタデータから分離されているAdobeのHTTP Dynamic Streaming（HDS）をサポートする必要があります。

解決法

`f4f`モジュールを介してフラグメント化されたFLVファイルのためのNGINX Plusのサポートを使用して、Adobe Adaptiveストリーミングをユーザーに提供します：

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

この例は、NGINX Plus `f4f`モジュールを使用して、以前フラグメント化されたメディアをディスク上の場所からクライアントに提供するようにNGINX Plusに指示します。インデックスファイル（*.f4x*）のバッファサイズは512 KBに設定されています。

解説

NGINX Plus f4fモジュールはNGINXが以前フラグメント化されたメディアをエンドユーザーに提供できるようにします。このような構成は、HTTPロケーションブロック内でf4fハンドラーを使用するのと同じくらい簡単です。f4f_buffer_sizeディレクティブは、このタイプのメディアのインデックスファイルのバッファサイズを設定します。

9.4 NGINX Plusの帯域幅制限

問題

視聴体験に影響を与えることなく、ダウンストリームのメディアストリーミングクライアントへの帯域幅を制限する必要があります。

解決法

NGINX PlusのMP4メディアファイルのビットレート制限サポートを利用します：

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

この構成により、ダウンストリームクライアントはビットレート制限を適用する前に15秒間ダウンロードできます。15秒後、クライアントはビットレートの120%のレートでメディアをダウンロードできるようになります。これにより、クライアントは常に再生よりも速くダウンロードできるようになります。

解説

NGINX Plusのビットレート制限により、ストリーミングサーバーは、提供されるメディアに基づいて帯域幅を動的に制限できるため、クライアントはシームレスなユーザーエクスペリエンスを確実に提供するのに必要なだけダウンロードできます。レシピ9.1で説明されているMP4ハンドラーは、このロケーションブロックを指定してMP4メディア形式をストリーミングします。mp4_limit_rate_afterなどのレート制限ディレクティブは、指定された時間（秒単位）経過後にのみトラフィックをレート制限するようにNGINXに指示します。MP4レート制限に関係する他のディレクティブはmp4_limit_rateです。これは、メディアのビットレートに関連して、クライアントがダウンロードを許可されるビットレートを指定します。mp4_limit_rateディレクティブに指定された値1は、NGINXが帯域幅（1対1）をメディアのビットレートに制限するよう指定します。mp4_limit_rateディレクティブに1より大きい値を指定すると、ユーザーは視聴するよりも速くダウンロードできるため、ダウンロード中にメディアをバッファできるためシームレスに視聴できます。

第10章
クラウド展開

10.0 はじめに

クラウドプロバイダーは、Webアプリケーションホスティングの状況を変えました。新しいマシンのプロビジョニングなど、以前は数時間から数か月かかっていたプロセスが、クリックまたはAPI呼び出しだけで作成できるようになりました。これらのクラウドプロバイダーは、インフラストラクチャ・アズ・ア・サービス（IaaS）と呼ばれる仮想マシン、またはデータベースなどのマネージドソフトウェアソリューションを、従量課金モデルを通じてリースし、ユーザーは使用した分だけ料金を支払います。これにより、エンジニアはすぐにテスト用の環境全体を構築し、不要になったら、破棄することができます。クラウドプロバイダーを使用すると、アプリケーションをパフォーマンスのニーズに基づいて即座に水平方向に拡張することもできます。本章では、主要なクラウドプロバイダープラットフォームでの基本的なNGINXおよびNGINX Plusのデプロイについて説明します。

10.1 自動プロビジョニング

問題

マシンが自動的にプロビジョニングするように、Amazon Web Services（AWS）でのNGINX サーバーの構成を自動化する必要があります。

解決法

このセクションでは、AWSを例に説明しますが、自動プロビジョニングに関するコアコンセプトは、Azure、Google Cloud Platform（GCP）、DigitalOceanなどの他のクラウドプロバイダーにも適用されます。

Amazon Elastic Compute Cloud（EC2）`UserData`とプリベイクされたAmazon Machine Image（AMI）を利用します。NGINXとサポートするソフトウェアパッケージがインストールされたAMIを作成します。EC2 `UserData`を使用して、環境固有の構成をランタイムで構成します。

解説

AWSでプロビジョニングする場合、3つの考え方があります：

ブート時にプロビジョン

一般的なLinuxイメージから開始し、起動時に構成管理またはシェルスクリプトを実行して、サーバーを構成します。このパターンはスタートが遅く、エラーが発生しやすい傾向があります。

AMIの作成

サーバーを完全に構成してから、AMIを焼きます。このパターンは起動が非常に速く正確です。しかし、周辺環境に対する柔軟性に欠け、多くのイメージの保守が複雑になる可能性があります。

AMIの一部を作成する

上記2つの方法の合わせ技です。部分的に作成されるのは、ソフトウェア要件がインストールされてAMIに書き込まれる場所であり、環境構成は起動時に行われます。このパターンは、完全にAMI作成されたパターンと比較して柔軟性があり、起動時のプロビジョニングソリューションと比較して高速です。

AMIを一部または完全に作成することのどちらを選んだ場合でも、そのプロセスを自動化する必要があります。AMIビルドパイプラインを構築するために、いくつかのツールの使用が提案されています：

構成管理

構成管理ツールは、NGINXのどのバージョンを実行するか、どのユーザーとして実行するか、どのDNSリゾルバを使用するか、誰にアップストリームをプロキシするかなど、サーバーの状態をコードで定義します。この構成管理コードは、ソフトウェアプロジェクトのように、ソースのコントロールおよびバージョンの管理を実行できます。一般的な構成管理ツールとしては、ChefおよびAnsibleなどが挙げられます。これらは第5章で説明されています。

HashiCorpのPacker

Packerは、ほぼすべての仮想環境またはクラウドプラットフォームでの構成管理の実行を自動化し、実行が成功した場合にマシンイメージを書き込みするために使用されます。Packerは基本的に、選択したプラットフォーム上に仮想マシン（VM）を構築し、VMにSSHで接続し、指定したプロビジョニングを実行して、イメージを書き込みます。Packerを使用して、構成管理ツールを実行し、マシンイメージを仕様に合わせて確実に書き込むことができます。

起動時に環境設定をプロビジョニングするには、Amazon EC2 `UserData`を使用して、インスタンスが最初に起動された時にコマンドを実行できます。一部AMI作成処理された方法を使用している場合は、これを利用して、起動時に環境ベースのアイテムを構成できます環境ベースの構成の例としては、リッスンするサーバー名、使用するリゾルバ、プロキシ先のドメイン名、最初のアップストリームサーバープールなどがあります`UserData`は、base64でエンコードされた文字列であり、最初の起動時と実行時にダウンロードされます。`UserData`は、AMI内の他

のブートストラッププロセスによってアクセスされる環境ファイルのように単純な場合もあれば、AMIに存在する任意の言語で記述されたスクリプトの場合もあります。UserDataは、構成管理に渡すために、変数を指定する、または変数をダウンロードするバッシュスクリプトであることが一般的です。構成管理により、システムが正しく構成され、環境変数に基づいて構成ファイルがテンプレート化され、サービスがリロードされます。UserDataの実行後、非常に信頼性の高い方法でNGINXマシンを完全に構成する必要があります。

10.2 NGINX VMのクラウドへのデプロイ

問題

クラウドプロバイダーでNGINXサーバーを作成して、リソースのロードバランスまたはプロキシを行う必要があります。

解決法

このセクションでは、Azureを例に説明しますが、NGINX VMのデプロイに関するコアコンセプトは、AWS、GCP、DigitalOceanなどの他のクラウドプロバイダーにも適用されます。

コンソールの[Azure仮想マシン]セクション内で新しいVMを作成します。Azureサブスクリプションとリソースグループを選択するプロンプトが表示されます。リソースグループがない場合、オプションを使用して作成してください。VMの名前、[リージョン]、[アベイラビリティオプション]、[セキュリティタイプ]、[ベースイメージ]を指定します。最も使い慣れたLinuxディストリビューションを選択します。さらに、要求されるVMのサイズ、およびAzureによる管理者アカウントの設定方法についての情報を提供する必要があります。[インバウンドポート]セクションで、SSH経由でアクセスできるようにポート22が開いていることを確認します。[次へ]ボタンをクリックして、[ディスク構成]セクションに移動します。

ワークロードに最適なディスクの構成オプションを入力します。NGINXを静的アセットの提供やリソースのキャッシュに使用する場合は、ディスクのサイズと速度を考慮して[ネットワーキング]セクションに進みます。

[ネットワーキング]セクションで、VM用の新しい仮想ネットワークとサブネットを選択または作成します。このVMにパブリックインターネットからアクセスできるようにパブリックIPを割り当て、SSHアクセスのためのポート22が開いていることを確認します。クリーンアップを簡単にするために、[VMが削除されたときにパブリックIPとNICを削除する]オプションを選択することをお勧めします。

特別な理由がない限り、残りのセクションの設定はデフォルトのままにしてください。[確認および作成]ボタンを選択します。構成を確認して[作成]ボタンをクリックし、Azureコンソールがデプロイの進捗画面に進むまで待ちます。

Azureは進行状況を表示しながら、必要なリソースをすべて作成および構成します。すべてのリソースが作成されると、[リソースに移動]ボタンが青くなります。このボタンをクリックすると、新しく作成したVMの情報が表示されます。

[プロパティ]タブの[ネットワーキング]セクションで、VMのパブリックIPを確認します。このIPと、構成中に提供された管理者認証情報を使用して、VMにSSHアクセスします。特定のOSタイプのパッケージマネージャーを介してNGINXまたはNGINX Plusをインストールします。適宜NGINXを構成してリロードします。もしくは、カスタムデータ起動スクリプトを使用してNGINXをインストールして構成することもできます。これは、VMを作成するときの[詳細]セクションの構成オプションです。

解説

Azureは、仮想化されたクラウド環境でのネットワーキングとコンピューティングとともに、高度に構成可能なVMを即座に提供します。VMの起動は簡単で、無限の可能性が広がります。Azure VMを使用すると、いつでもどこでも必要な時にNGINXサーバーの全機能を利用できます。

10.3 NGINXマシンイメージの作成

問題

VMをすばやくインスタンス化するか、インスタンスグループのインスタンステンプレートを作成するために、NGINXマシンイメージを作成する必要があります。

解決法

このセクションでは、GCPを例に説明しますが、マシンイメージの作成に関するコアコンセプトは、AWS、Azure、DigitalOceanなどの他のクラウドプロバイダーにも適用されます。

VMインスタンスにNGNXをインストール、構成後、ブートディスクの自動削除状態をfalseに設定します。ディスクの自動削除状態を設定するには、VMを編集します。ディスク構成の下の[編集]ページには、[インスタンスが削除されたらブートディスクを削除する]というチェックボックスがあります。このチェックボックスの選択を解除して、VM構成を保存します。インスタンスの自動削除状態がfalseに設定されたら、インスタンスを削除します。メッセージが表示されても、ブートディスク削除のチェックボックスを選択しないでください。これらのタスクを実行すると、NGINXがインストールされたアタッチされていないブートディスクが残ります。

インスタンスが削除され、ブートディスクが接続されていない場合は、Google Compute Imageを作成できます。Google Compute Engineコンソールの［イメージ］セクションで、［イメージの作成］を選択します。画像名、ファミリ、説明、暗号化タイプ、ソースの入力を求められます。使用する必要のあるソースタイプはディスクです。使用するソースのタイプはディスクです。ソースディスクには、接続されていないNGINXブートディスクを選択します。［作成］を選択すると、Google Compute Cloudがディスクからイメージを作成します。

解説

Google Cloud Imageを利用して、作成したサーバーと同じブートディスクでVMを作成できます。イメージを作成することの価値は、このイメージのすべてのインスタンスが同一であることを保証できる点です。動的環境で起動時にパッケージをインストールする場合には、プライベートリポジトリでバージョンロックを使用しない限り、本番環境で実行する前にパッケージのバージョンと更新が検証されない恐れがあります。マシンイメージを使用すると、このマシンで実行されているすべてのパッケージがテストしたとおりであることを検証でき、提供するサービスの信頼性が強化されます。

関連項目

Google Cloud Images：カスタムイメージの作成ドキュメンテーション

10.4 クラウドネイティブロードバランサーを使用しないNGINXノードへのルーティング

問題

トラフィックを複数のアクティブなNGINXノードにルーティングするか、アクティブ-パッシブのフェイルオーバーセットを作成して、NGINXの前にロードバランサーがない状態で高可用性を実現する必要があります。

解決法

このセクションでは、AWSを例に説明しますが、DNS Aレコードに関するコアコンセプトは、Azure、GCP、DigitalOceanなどの他のクラウドプロバイダーにも適用されます。Amazon Route 53 DNSサービスを使用して、複数のアクティブなNGINXノードにルーティングするか、ヘルスチェックとNGINXノードのアクティブ-パッシブセット間のフェイルオーバーを構成します。

解説

DNSは、長い間サーバー間の負荷を分散してきました。クラウドに移行しても、それは変わりません。AmazonのRoute 53サービスは、多くの高度な機能を備えたDNSサービスを提供し、すべてAPIを介して利用できます。単一のAレコード上の複数のIPアドレスや重み付きAレコードなど、すべての一般的なDNS使用法を利用できます。複数のアクティブなNGINXノードを

実行している場合は、これらのAレコード機能のいずれかを使用して、すべてのノードに負荷を分散したいと考えることでしょう。ラウンドロビンアルゴリズムは、単一のAレコードに複数のIPアドレスがリストされている場合に使用されます。重み付き分散を使用すると、Aレコードで各サーバー IPアドレスの重みを定義することにより、負荷を不均一に分散できます。

Route 53のより興味深い機能はヘルスチェックです。TCP接続を確立するか、HTTPまたはHTTPSでリクエストを行うことにより、エンドポイントの状態を監視するようにRoute 53を設定できます。ヘルスチェックは、IP、ホスト名、ポート、URIパス、間隔レート、監視、および地理のオプションを使用して非常に柔軟に構成可能です。これらのヘルスチェックを使用すると、Route53は、ヘルスチェックに失敗し始めた場合に、IPをローテーションから外すことができます。また、障害が発生した場合にセカンダリレコードにフェイルオーバーするようにRoute 53を設定することもできます。これにより、アクティブ-パッシブ、高可用性のセットアップが実現します。

Route 53には、地理ベースのルーティング機能があり、クライアントを最も近いNGINXノードにルーティングして待ち時間を最小限に抑えることができます。地理的にルーティングする場合、クライアントは最も近い健全な物理的ロケーションに転送されます。アクティブ-アクティブ構成で複数のインフラストラクチャセットを実行している場合、ヘルスチェックを使用して別の地理的ロケーションに自動的にフェイルオーバーできます。

Route 53 DNSを使用してトラフィックをAuto ScalingグループのNGINXノードにルーティングする場合は、DNSレコードの作成と削除を自動化する必要があります。NGINXノードのスケーリングに合わせてRoute 53へのNGINXマシンの追加と削除を自動化するには、Amazon Auto Scalingライフサイクルフックを使用して、NGINXボックス自体のスクリプトをトリガーするか、またはAmazon Lambdaで独立して実行されるスクリプトをトリガーできます。これらのスクリプトは、AmazonCommandLineInterface (CLI) またはソフトウェア開発キット（SDK）を使用してAmazon Route 53 APIとインターフェースし、NGINXマシンのIPを追加または削除し、起動時または終了前にヘルスチェックを構成します。

関連項目

Amazon Route 53およびNGINX Plusグローバルサーバーロードバランシング

ブログ（英語）：“Set Up Dynamic DNS for AWS EC2 Instance with Lambda Service”

10.5 ロードバランサーサンドイッチ

問題

NGINXオープンソースレイヤーを自動スケーリングし、アプリケーションサーバー間で負荷を均一かつ簡単に分散する必要があります。

解決法

このセクションでは、AWSを例に説明しますが、クラウドネイティブロードバランサー間でのNGINXのサンドイッチに関するコアコンセプトは、Azure、GCP、DigitalOceanなどの他のクラウドプロバイダーにも適用されます。NLBが提供する機能セットはNGINX Plusで利用できるため、このパターンはNGINXオープンソースにのみ必要です。

ネットワークロードバランサー（NLB）を作成します。コンソールを介したNLB の作成中に、新しいターゲットグループを作成するよう求められます。コンソールを介して行わない場合は、このリソースを作成して、NLB のリスナーにアタッチする必要があります。NGINX オープンソースがインストールされたEC2 インスタンスをプロビジョニングするローンチ設定でAuto Scalingグループを作成します。Auto Scalingグループには、ターゲットグループにリンクする構成があり、Auto Scalingグループ内のすべてのインスタンスが、初回起動時に構成されたターゲットグループに自動的に登録されます。ターゲットグループは、NLB のリスナーによって参照されます。アップストリームアプリケーションを別のネットワークロードバランサーとターゲットグループの後ろに配置して、その後、アプリケーションNLB にプロキシするようにNGINX を構成します。

解説

この一般的なパターンはロードバランサーサンドイッチ（図10-1参照）と呼ばれ、NGINXをNLBの背後にあるAuto Scalingグループに配置し、アプリケーションのAuto Scalingグループを別のNLBの背後に配置します。各レイヤーの間にNLBを置く理由は、NLBがAuto Scalingグループで効果的に機能するためです。NLBは、新しいノードを自動的に登録し、既存ノードを削除し、ヘルスチェックを実行し、正常なノードのみにトラフィックを渡します。

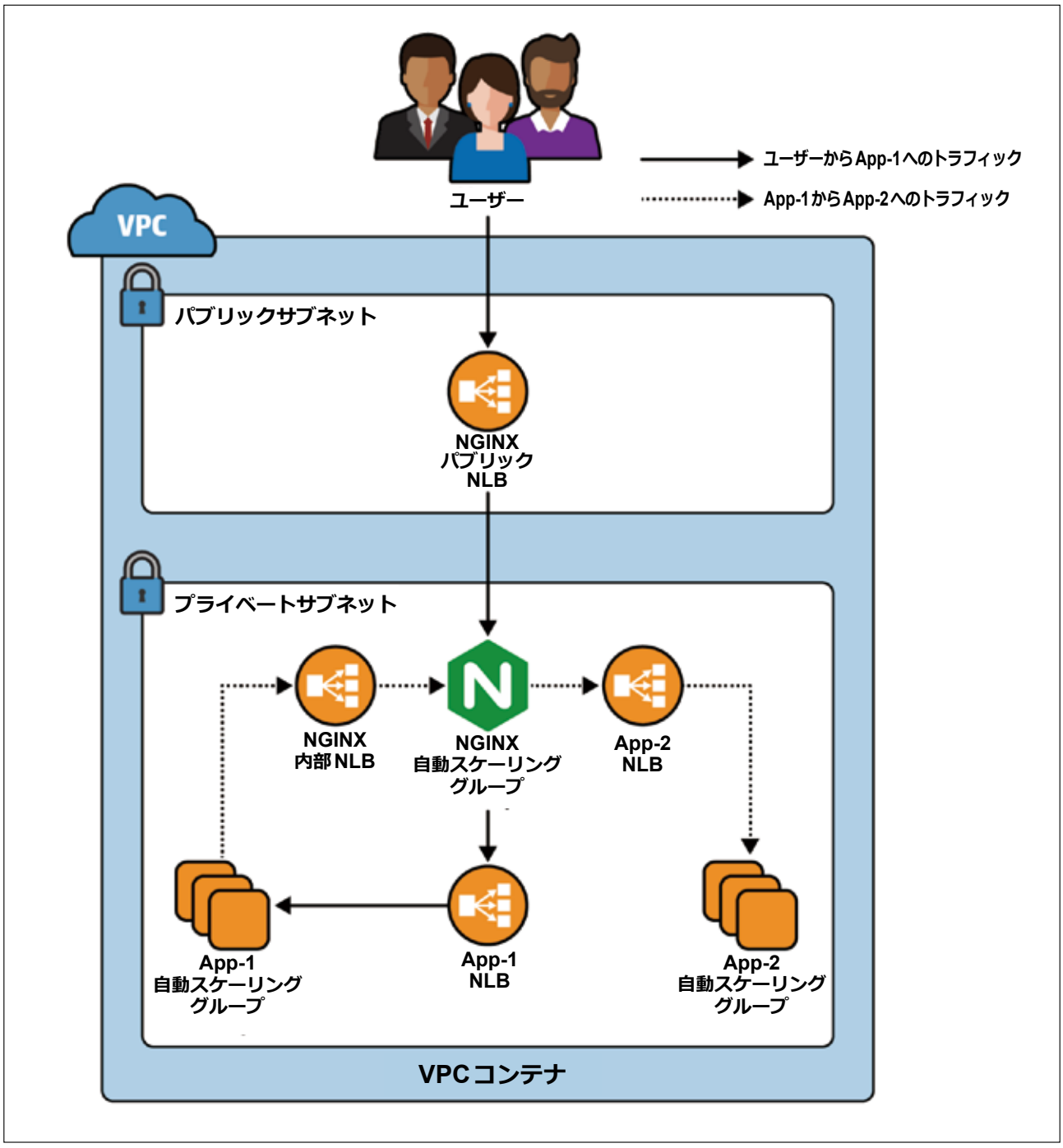


図10-1　この図はロードバランサーサンドイッチパターンのNGINXと内部アプリケーションが利用できる内部NLBを示しています。ユーザーがApp-1にリクエストを送信し、App-1がNGINXを介してApp-2にリクエストを送信して、ユーザーのリクエストを実行します。

NGINXオープンソースレイヤーのために2つ目の内部NLBを構築する必要があるかもしれません。これにより、アプリケーション内のサービスは、ローカルのネットワークの外側へ出て、パブリックのNLBを通って再度接続することなく、NGINX Auto Scalingグループを通じて他のサービスを呼び出すことができます。これにより、NGINXはアプリケーション内のすべてのネットワークトラフィックの中央に配置され、アプリケーションのトラフィックルーティングの中心になります。このパターンは、以前は*Elasticロードバランサー（ELB）サンドイッチ*と呼ばれていました。しかし、NLBはレイヤー4のロードバランサーであるのに対し、ELBとALBはレイヤー7のロードバランサーであるため、NGINXを使用する場合はNLBが推奨されます。レイ

ヤー 7ロードバランサーはPROXYプロトコルを介してリクエストを変換し、NGINXを使用して冗長化されます。

このパターンは、他のクラウドプロバイダーでも同様に使用できます。Azureロードバランサーとスケールセット、またはGCPロードバランサーとAuto Scalingグループを使用している場合でも、コンセプトは同じです。このパターンの核となる価値は、スケーリングするアプリケーションサーバーの自動登録とロードバランシングを提供するためにクラウドネイティブサービスを使用し、プロキシロジックにNGINXを使用することです。

10.6 動的にスケーリングするNGINXサーバーのロードバランシング

問題

クラウドネイティブロードバランサーの背後にあるNGINXノードをスケーリングして、高可用性と動的なリソース使用を行う必要があります。

解決法

このセクションでは、Azureを例に説明しますが、NGINXレイヤーのスケーリングおよびそこへのトラフィックのルーティング方法に関するコアコンセプトは、AWS、GCP、DigitalOceanなどの他のクラウドプロバイダーにも適用されます。

公開または内部のAzureロードバランサーを作成します。NGINX VMイメージ、またはマーケットプレースのNGINX PlusイメージをAzure仮想マシンスケールセット（VMSS）にデプロイします。ロードバランサーとVMSSがデプロイされたら、ロードバランサーのバックエンドプールをVMSSに構成します。トラフィックを受け入れるポートとプロトコルのロードバランスルールをセットアップし、それらをバックエンドプールに転送します。

解説

NGINXをスケーリングして高可用性を実現したり、リソースを過剰にプロビジョニングせずにピーク時の負荷を処理したりするのが一般的です。Azureでは、VMSSを使用してこれを実現します。Azureロードバランサーを使用すると、スケーリング時にリソースのプールにNGINXノードを追加、削除するための管理が容易になります。Azureロードバランサーを使用すると、バックエンドプールの状態を確認し、正常なノードにのみトラフィックを渡すことができます。内部ネットワーク経由でのみアクセスを有効にするNGINXの前で内部Azureロードバランサーを実行できます。NGINXを使用して、VMSS内のアプリケーションの前にある内部ロードバランサーにプロキシできます。ロードバランサーを使用すると、プールへの登録と登録解除が簡単になります。

10.7 Google App Engineプロキシの作成

問題

アプリケーション間でコンテキストスイッチしたり、カスタムドメインでHTTPSを提供したりするために、Google AppEngineのプロキシを作成する必要があります。

解決法

Google Compute CloudでNGINXを利用します。Google Compute EngineでVMを作成するか、NGINXがインストールされたVMイメージを作成し、このイメージをブートディスクとして使用してインスタンステンプレートを作成します。インスタンステンプレートを作成した場合は、そのテンプレートを利用するインスタンスグループを作成してフォローアップします。

GoogleAppEngineエンドポイントにプロキシするようにNGINXを設定します。GoogleApp Engineは公開されているため、必ずHTTPSにプロキシするようにします。また、NGINXインスタンスでHTTPSを終了せず、NGINXとGoogle AppEngineの間をセキュリティで保護されていない状態で情報をやり取りできるようにする必要があります。App Engineでは単一のDNSエンドポイントのみが提供されるので、NGINXオープンソースを使用する場合は`upstream`ブロックではなく`proxy_pass`ディレクティブを使用する必要があります。アップストリームのDNS名は、NGINX Plusでのみ使用できます。Google App Engineにプロキシするときは、必ずエンドポイントをNGINXの変数として設定してから、`proxy_pass`ディレクティブでその変数を使用して、NGINXがすべての要求でDNS解決を行うようにします。NGINXでDNS解決を行うには、`resolver`ディレクティブも利用して、お気に入りのDNSリゾルバーをポイントする必要があります。Googleは、IPアドレス8.8.8.8をパブリックに使用できるようにしています。NGINX Plusを使用している場合、Google App Engineにプロキシするときに、`upstream`ブロック内の`server`ディレクティブで`resolve`フラグ、キープアライブ接続、アップストリームモジュールのその他の利点を使用できます。

NGINX構成ファイルをGoogleクラウドストレージに保存し、インスタンスの起動スクリプトを使用して、起動時に構成をプルダウンすることを選択できます。これにより、新しいイメージを書き込むことなく構成を変更できます。しかし、NGINXサーバーのスタートアップ時間が延びます。

解説

独自のドメインを使用していて、HTTPS経由でアプリケーションを利用できるようにする場合は、Google App Engineの前でNGINXを実行する必要があります。この時点では、Google App Engineでは独自のSSL証明書をアップロードすることはできません。したがって、暗号化を使用してGoogle提供のappspot.com以外のドメインでアプリを提供する場合は、カスタムドメインでリッスンするためにNGINXでプロキシを作成する必要があります。NGINXは、NGINXとクライアント間、およびNGINXとGoogle App Engine間の通信を暗号化します。

Google App Engineの前でNGINXを実行するもう1つの理由は、同じドメインで多数のApp Engineアプリをホストし、NGINXを使用してURIベースのコンテキスト切り替えを行うためで

す。マイクロサービスは人気のあるアーキテクチャであり、NGINXのようなプロキシがトラフィックルーティングを実行するのが一般的です。Google App Engineを使用すると、アプリケーションを簡単にデプロイできます。NGINXと組み合わせることで、本格的なアプリケーションデリバリープラットフォームを利用できます。

第11章
コンテナ/マイクロサービス

11.0 はじめに

コンテナは、アプリケーション層で抽象化の層を提供し、パッケージのインストールと依存関係をデプロイからビルドプロセスにシフトします。エンジニアは現在、環境に関係なく統一された方法で実行/展開されるコードのユニットを送信しているため、これは重要ですコンテナを実行可能なユニットとして昇格させると、環境間の依存関係や構成の障害のリスクが軽減されます。こうした状況に基づき、組織がコンテナプラットフォームにアプリケーションを展開することには、大きな推進力がありました。コンテナプラットフォームでアプリケーションを実行する場合、プロキシやロードバランサーを含め、できるだけ多くのスタックをコンテナ化するのが一般的です。NGINXは簡単にコンテナ化とデプロイを行います。また、コンテナ化されたアプリケーションの配信を流動的にする多くの機能も含まれています。本章では、NGINXコンテナイメージの構築、コンテナ化された環境での作業を容易にする機能、そして、イメージをKubernetesとOpenShiftにデプロイすることを中心に説明します。

コンテナ化する場合、サービスをより小さなアプリケーションに分解するのが一般的です。これを実行する時、APIゲートウェイによって一緒に結び付けられます。本章では、NGINXをAPIゲートウェイとして使用して、リクエストを保護、検証、認証し、適切なサービスにルーティングする一般的なケースシナリオを示します。

コンテナでのNGINXの実行に関するアーキテクチャ上の考慮事項を確認する必要があります。サービスをコンテナ化する場合、Dockerのログドライバーを使用するには、ログを*/dev/stdout*に出力し、エラーログを*/dev/stderr*に送信する必要があります。 こうすることで、ログはDockerログドライバーにストリームされ、Dockerログドライバーはそれらを統合ログサーバーにネイティブにルーティングできます。

コンテナ化された環境でNGINX Plusを使用する場合は、ロードバランシング方法も考慮する必要があります。`least_time`ロードバランシングメソッドはコンテナ化されたネットワークオーバーレイ を念頭に置いて設計されました。NGINX Plusは、応答時間を短くすることで、平均応答時間が最も速いアップストリームサーバーに要求を渡します。すべてのサーバーが適切にロードバランスされ、同等のパフォーマンスを発揮する場合、NGINX Plusはネットワーク遅延を最適化し、ネットワークに最も近いサーバーを優先します。

11.1 APIゲートウェイとしてのNGINXの使用

問題

ユースケースでは、受け取った要求を検証、認証、操作、ルーティングするためにAPIゲートウェイを利用したいです。

解決法

NGINXをAPIゲートウェイとして使用します。APIゲートウェイは、1つ以上のAPIへのエントリポイントを提供します。NGINXはこの役割にとてもよく適合します。このセクションでは、いくつかのコアコンセプトをハイライトし、詳細については、本書のその他の部分を参照します。また、NGINXはこのトピックに関する電子書籍（ebook）API ゲートウェイとしてのNGINXのデプロイ（Liam Crilly著）も公開していますのでご覧ください。

固有ファイル内でAPIゲートウェイのサーバーブロックを定義することから始めます。*/etc/nginx/api_gateway.conf*などの名前で良いでしょう。

```
server {
  listen 443 ssl;
  server_name api.company.com;
  # SSL Settings Chapter 7

  default_type application/json;
}
```

サーバー定義にいくつかの基本的なエラー処理応答を追加します：

```
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'; }
```

NGINX構成の上記のセクションは、*/etc/nginx/api_gateway.conf*のサーバーブロックまたは別のファイルに直接追加し、`include`ディレクティブを介してインポートできます。`include`ディレクティブについてはレシピ1.6で説明されています。

`include`ディレクティブを使用して、このサーバー構成を`http`コンテキスト内のメイン*nginx.conf*ファイルにインポートします：

```
include /etc/nginx/api_gateway.conf;
```

次に、アップストリームサービスエンドポイントを定義する必要があります。第2章はロードバランシングについて説明します。upstreamブロックについても触れられています。upstreamはhttpコンテキスト内で有効ですが、serverコンテキスト内では有効ではありませんので注意が必要です。 以下を含めるか、serverブロックの外に設定する必要があります：

```
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;
}
```

ユースケースに応じて、サービスをインラインでインクルードファイルとして宣言するか、サービスごとのインクルードとして宣言できます。サービスをプロキシロケーションエンドポイントとして定義する必要がある場合もあります。この場合、エンドポイントを全体で使用する変数として定義することをお勧めします。第5章「プログラマビリティと自動化」は、upstreamブロックからのマシンの追加と削除を自動化する方法について説明しています。

各サービスのserverブロック内に内部的にルーティング可能な場所を構築します。

```
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;
}
```

これらのサービスの内部ルーティング可能な場所を定義することにより、サービスに共通の構成を繰り返しではなく1回定義できます。

ここから、特定のサービスの固有のURIパスを定義するlocationブロックを構築する必要があります。これらのブロックは、リクエストを検証し、適切にルーティングします。. APIゲートウェイは、パスに基づいてリクエストをルーティングするのと同じくらい単純でありながら、受け入れられるすべてのAPI URIに特定のルールを定義するのと同じくらい詳細に定義できます。後者の場合、組織のためのファイル構成を考案し、NGINXのincludeを使用して設定ファイルをインポートしたいと思う場合があります。このコンセプトはレシピ1.6で説明されています。

APIゲートウェイのために新しいディレクトリを作成します：

```
mkdir /etc/nginx/api_conf.d/
```

構成の構造に適したパスでファイル内のlocation ブロックを定義し、サービ構成構造にとって

意味のあるパスにあるファイル内でロケーション ブロックを定義することにより、サービスユースケースの仕様を構築します。rewriteディレクティブを使用して、リクエストをサービスにプロキシする以前に構成されたlocationブロックにリクエストを送信します。以下の例で使用されるrewriteディレクティブは、変更されたURIを使用してリクエストを再処理するようにNGINXに指示します。この例では、APIリソースに特定のルールを定義し、HTTPメソッドを制限し、サービスのために事前に定義された内部共通のプロキシロケーションにリクエストを送信するため、rewriteディレクティブを使用します：

```
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;
}
```

各サービスでこの手順を繰り返します。ファイルとディレクトリ構造による論理的分離を使用して、ユースケースを効果的に整理します。本書で提供されているすべての情報を使用して、APIlocationブロックを可能な限り具体的かつ制限的に構成します。

上記のlocationまたはupstreamブロックに個別のファイルが使用されている場合は、それらがserverコンテキストでincludeされていることを確認してください：

```
server {
  listen 443 ssl;
  server_name api.company.com;
  # SSL Settings Chapter 7

  default_type application/json;

  include api_conf.d/*.conf;
}
```

プライベートなリソースを保護するために認証を有効にするには、第6章で説明した多くの方法のうちのひとつを使用するか、あるいは以下のように事前共有APIキーのような単純なものを使用します(mapディレクティブはhttpコンテキストでのみ有効なことに注意してください)。

```
map $http_apikey $api_client_name {
  default "";

  "j7UqLLB+yRv2VTCXXDZ1M/N4" "client_one";
  "6B2kbyrrTiIN8S8JhSAxb63R" "client_two";
  "KcVgIDSY4Nm46m3tXVY3vbgA" "client_three";
}
```

NGINXでバックエンドサービスを攻撃から守るには、第2章から学んだことを活かして使用量を制限します。httpコンテキストで、1つまたは複数のリクエスト制限共有メモリゾーンを定義します。

```
limit_req_zone $http_apikey
  zone=limitbyapikey:10m rate=100r/s;
limit_req_status 429;
```

特定のコンテキストをレート制限と認証で保護します：

```
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;
}
```

APIゲートウェイへの呼び出しをテストします：

```
$ curl -H "apikey: 6B2kbyrrTiIN8S8JhSAxb63R" \
    https://api.company.com/api/service_2/object
```

解説

APIゲートウェイは、APIへのエントリポイントを提供します。では、この点について詳しく見ていきましょう。統合ポイントは、さまざまなレイヤーで発生します。通信（統合）する必要のある2つの独立したサービスは、APIバージョンコントラクトを保持する必要があります。このようなバージョンコントラクトは、サービスの互換性を定義します。APIゲートウェイは、サービス間のリクエストの認証、承認、変換、ルーティングなどのコントラクトを実施します。

このセクションでは、受信したリクエストを検証、認証、特定のサービスに送信し、それらの使用を制限することで、NGINXがAPIゲートウェイとして機能する方法を示しました。この方法は、単一のAPIオファリングが複数の異なるサービスに分割されるマイクロサービスアーキテクチャで人気があります。

これまでに学んだことを実装して、ユースケースの仕様に合わせてNGINXサーバー構成を構築してください。このテキストで示されているコアコンセプトを組み合わせることで、URIパスの使用を認証および承認したり、任意の要因に基づいてリクエストをルーティングまたは書き換えたりするほか、使用を制限し、有効なリクエストとして受け入れられるものと受け入れられないものを定義することができます。APIゲートウェイは、提供するユースケースに密接に関連し、無限に定義できるため、APIゲートウェイに対する唯一のソリューションというものはありません。

APIゲートウェイは、運用チームとアプリケーションチームの間に究極のコラボレーションスペースを提供し、真のDevOps組織を形成できるようにします。アプリケーション開発では、特定のリクエストの有効なパラメータを定義します。こうしたリクエストの配信は、通常、ITと見なされるもの（ネットワーク、インフラストラクチャ、セキュリティ、およびミドルウェアのチーム）によって管理されます。 APIゲートウェイは、これら2つのレイヤー間のインターフェースとして機能します。APIゲートウェイの構築には、すべての関係者からのインプットが必要です。その構成は、ある種のソース管理の範囲の中で行われる必要があります。現代の多くのソース管理リポジトリには、コード所有者のコンセプトがあります。このコンセプトにより、特定のファイルに対して特定のユーザーの承認を要求できます。この方法で、チームは共同作業を行いながら、特定の部門に固有の変更を検証することができます。

APIゲートウェイの使用にあたり留意すべき点は、URIパスです。この構成例では、URIパス全体がアップストリームサーバーに渡されます。つまり、`service_1`例は*/api/ service_1/**パスにハンドラーが必要だということです。この方法でパスベースのルーティングを実行するには、アプリケーションに別のアプリケーションと競合するルートがないことがベストです。

競合するルートが適用される場合にできることがいくつかあります。コードを編集して競合を解決するか、一方または両方のアプリケーションにURIプレフィックス構成を追加して、一方を別のコンテキストに移動します。編集できない既製のソフトウェアの場合は、リクエストURIアップストリームを書き換えることができます。ただし、アプリケーションが本文にリンクを記載して返す場合は、クライアントに提供する前に、正規表現（regex）を使用してリクエストの本文を書き換える必要があります。これは避けるべきやり方です。

関連項目

APIゲートウェイとしてNGINXをデプロイ（ebook）

11.2 NGINX PlusでのDNS SRVレコードの使用

問題

NGINX Plusを使用して、アップストリームサーバーのソースとして既存のDNSサービス（SRV）レコードの実装を使用したいと考えています。

解決法

アップストリームサーバーで`http`値を使用してサービスディレクティブを指定し、SRVレコードをロードバランシングプールとして利用するようにNGINXに指示します。

```
http {
  resolver 10.0.0.2 valid=30s;

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

```
    }
  }
```

この機能はNGINX Plusでのみ利用できます。この構成は、`10.0.0.2`のDNSサーバーからDNSを解決し、単一の`server`ディレクティブを使用してアップストリームサーバープールを設定するようにNGINX Plusに指示します。`resolve`パラメータで指定されたこの`server`ディレクティブは、DNSレコードTTL、または`resolver`ディレクティブの有効なオーバーライドパラメータに基づいてドメイン名を定期的に再解決するように指示されます。`service=http`パラメータと値は、これがIPとポートのリストを含むSRVレコードであり、`server`ディレクティブで構成されているかのようにそれらを負荷分散することをNGINXに通知します。

解説

動的インフラストラクチャは、クラウドベースのインフラストラクチャの需要と採用にともない、ますます人気が高まっています。Auto Scaling環境は水平方向にスケーリングし、負荷の需要に合わせてプール内のサーバーの数を増減します。水平方向にスケーリングするには、プールにリソースを追加したり、プールからリソースを削除したりできるロードバランサーが必要です。SRVレコードを使用すると、サーバーのリストをDNSに保持する責任を回避できます。このタイプの構成は、コンテナ化された環境にとって非常に魅力的です。コンテナが可変のポート番号で、場合によっては同じIPアドレスでアプリケーションを実行している可能性があるためです。UDP DNSレコードのペイロードは約512バイトに制限されている点にご注意ください。

11.3 公式のNGINXコンテナイメージの使用

問題

Docker HubのNGINXイメージを使用してすばやく起動して実行する必要があります。

解決法

Docker HubからNGINXイメージを使用します。このイメージにはデフォルトの構成が含まれます。構成を変更するには、ローカル構成ディレクトリをマウントするか、Dockerfileを作成して構成をイメージビルドにCOPYする必要があります。ここでは、NGINXのデフォルト構成が静的コンテンツを配信するボリュームをマウントし、単一のコマンドを使用してその機能を示します：

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

`docker`コマンドは、ローカルで見つからない場合、Docker Hubから`nginx:latest`イメージを取得します。次に、コマンドはこのNGINXイメージをDockerコンテナとして実行し、`localhost:80`をNGINXコンテナのポート`80`にマッピングします。また、ローカルディレクトリ*/path/to/content/*を*/usr/share/nginx/html/*にコンテナボリュームとして読み取り専用としてマウントします。デフォルトのNGINX構成は、このディレクトリを静的コンテンツとして提供します。ローカルマシンからコンテナへのマッピングを指定する場合、ローカルマシンのポート

またはディレクトリが最初になり、コンテナのポートまたはディレクトリは2番目になります。

解説

NGINXは、公式のコンテナイメージをDocker HubおよびAmazon Elastic Container Registryから利用できるようにしました。この公式のコンテナイメージを使用することで、お気に入りのアプリケーション配信プラットフォームであるNGINXを使用してDockerをすばやく簡単に起動できます。このセクションでは、たった1つのコマンドでNGINXをコンテナで起動して実行することができました。この例で使用した公式のNGINXコンテナイメージの主たる部分は、Debianに基づいて構築されています。ただし、Alpine Linuxに基づいた公式イメージや、最新コミットに対してメインラインから構築されたイメージ、Perlモジュールをインストールして構築されたイメージを選択できます。Dockerfileとこれらの公式イメージのソースは、GitHubで入手できます。独自のDockerfileを構築し、FROMコマンドで公式イメージを指定することで、公式イメージを拡張できます。NGINX構成ディレクトリをDockerボリュームとしてマウントして、公式イメージを変更せずにNGINX構成をオーバーライドすることもできます。

公式のNGINXコンテナイメージは、NGINXをルートユーザーとして実行します。OpenShiftのような非特権ユーザーがサービスを実行する必要があるプラットフォームで作業する場合、公式の非特権イメージである`nginxinc/nginx-unprivileged`を使用できます。

関連項目

公式のNGINXコンテナイメージ、NGINX

GitHubのDockerリポジトリ

NGINX非特権コンテナイメージ

11.4 NGINX Dockerfileの作成

問題

コンテナイメージを作成するために、NGINX Dockerfileを作成する必要があります。

解決法

F5は、Dockerfileを保守し、GitHubを介して配布します。これらのDockerfileに基づき独自のDockerfileを構築できます。これらのリポジトリには、NGINXのインストールに役立つスクリプトが多数含まれています。詳しくは、GitHubリポジトリを参照してください。

使い慣れたディストリビューションのコンテナイメージの`FROM`ラインから開始します。この例では、Debian 12を使用します。パッケージマネージャーが更新されていることを確認し、ディストリビューション用の公式のNGINXリポジトリを使用できるように、パッケージマネージャーのリポジトリと署名キーのインストールに必要な前提条件パッケージをインストールしてください。`COPY`コマンドを使用して、NGINX構成ファイルを追加します。`COPY`コマンドは、ローカルディレクトリのファイルをコンテナイメージに配置します。これを使用して、デフォ

ルトのNGINX構成を独自のものに置き換えます。オプションで、EXPOSEコマンドを使用して、指定されたポートを公開するようにDockerに指示するか、イメージをコンテナとして実行する際にこれを手動で実行します。イメージがコンテナとしてインスタンス化されたら、CMDを使用してNGINXを起動します。NGINXをフォアグランドで実行する必要があります。そのためには、NGINXを`-g "daemon off;"`で始めるか、構成に`daemon off;`を追加します。また、アクセスログの場合は*/dev/stdout*に、エラーログの場合は*/dev/stderr*にログを記録するようにNGINX構成を変更することもできます。こうすることで、ログがDockerデーモンに渡り、Dockerで使用するために選択したログドライバーに基づいて、ログをより簡単に利用できるようになります：以下に、NGINXにより提供されるDockerfileを示します：

```
FROM debian:bookworm-slim

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

ENV NGINX_VERSION 1.25.3
ENV NJS_VERSION 0.8.2
ENV PKG_RELEASE 1~bookworm

RUN set -x \
# create nginx user/group first, to be consistent throughout
# docker variants
  && groupadd --system --gid 101 nginx \
  && useradd --system --gid nginx --no-create-home --home \
  /nonexistent --comment "nginx user" --shell /bin/false \
  --uid 101 nginx \
  && apt-get update \
  && apt-get install --no-install-recommends --no-install-suggests \
  -y gnupg1 ca-certificates \
  && \
  NGINX_GPGKEY=573BFD6B3D8FBC641079A6ABABF5BD827BD9BF62; \
  NGINX_GPGKEY_PATH=/usr/share/keyrings/nginx-archive-keyring.gpg; \
  export GNUPGHOME="$(mktemp -d)"; \
  found=''; \
  for server in \
    hkp://keyserver.ubuntu.com:80 \
    pgp.mit.edu \
  ; do \
    echo "Fetching GPG key $NGINX_GPGKEY from $server"; \
    gpg1 --keyserver "$server" --keyserver-options timeout=10 \
    --recv-keys "$NGINX_GPGKEY" && found=yes && break; \
  done; \
  test -z "$found" && echo >&2 "error: failed to fetch GPG key \
  $NGINX_GPGKEY" && exit 1; \
  gpg1 --export "$NGINX_GPGKEY" > "$NGINX_GPGKEY_PATH" ; \
  rm -rf "$GNUPGHOME"; \
  apt-get remove --purge --auto-remove -y gnupg1 && \
```

```
  rm -rf /var/lib/apt/lists/* \
  && dpkgArch="$(dpkg --print-architecture)" \
  && nginxPackages=" \
    nginx=${NGINX_VERSION}-${PKG_RELEASE} \
    nginx-module-xslt=${NGINX_VERSION}-${PKG_RELEASE} \
    nginx-module-geoip=${NGINX_VERSION}-${PKG_RELEASE} \
    nginx-module-image-filter=${NGINX_VERSION}-${PKG_RELEASE} \
    nginx-module-njs=${NGINX_VERSION}+${NJS_VERSION}-${PKG_RELEASE} \
  " \
  && case "$dpkgArch" in \
    amd64|arm64) \
# arches officially built by upstream
      echo "deb [signed-by=$NGINX_GPGKEY_PATH] \
      https://nginx.org/packages/mainline/debian/ bookworm nginx" >> \
      /etc/apt/sources.list.d/nginx.list \
      && apt-get update \
      ;; \
    *) \
# we're on an architecture upstream doesn't officially build for
# let's build binaries from the published source packages
      echo "deb-src [signed-by=$NGINX_GPGKEY_PATH] \
      https://nginx.org/packages/mainline/debian/ \
      bookworm nginx" >> /etc/apt/sources.list.d/nginx.list \
      \
# new directory for storing sources and .deb files
      && tempDir="$(mktemp -d)" \
      && chmod 777 "$tempDir" \
# (777 to ensure APT's "_apt" user can access it too)
      \
# save list of currently installed packages to build
# dependencies can be cleanly removed later
      && savedAptMark="$(apt-mark showmanual)" \
      \
# build .deb files from upstream's source packages
# (which are verified by apt-get)
      && apt-get update \
      && apt-get build-dep -y $nginxPackages \
      && ( \
        cd "$tempDir" \
        && DEB_BUILD_OPTIONS="nocheck parallel=$(nproc)" \
          apt-get source --compile $nginxPackages \
      ) \
# we don't remove APT lists here because they get redownloaded
# and removed later
      \
# reset apt-mark's "manual" list so that "purge --auto-remove"
# will remove all build dependencies
```

```
    # (which is done after we install the built packages so we
    # don't have to redownload any overlapping dependencies)
          && apt-mark showmanual | xargs apt-mark auto > /dev/null \
          && { [ -z "$savedAptMark" ] || apt-mark manual $savedAptMark; } \
          \
    # create a temporary local APT repo to install from
    # (so that dependency resolution can be handled by APT, as it should be)
          && ls -lAFh "$tempDir" \
          && ( cd "$tempDir" && dpkg-scanpackages . > Packages ) \
          && grep '^Package: ' "$tempDir/Packages" \
          && echo "deb [ trusted=yes ] file://$tempDir ./" > \
               /etc/apt/sources.list.d/temp.list \
    # work around the following APT issue by using
    # "Acquire::GzipIndexes=false" (overriding
    # "/etc/apt/apt.conf.d/docker-gzip-indexes")
    # Could not open file
    # /var/lib/apt/lists/partial/_tmp_tmp.ODWljpQfkE_._Packages -
    # open (13: Permission denied)
    # ...
    # E: Failed to fetch
    # store:/var/lib/apt/lists/partial/_tmp_tmp.ODWljpQfkE_._Packages
    # Could not open file
    # /var/lib/apt/lists/partial/_tmp_tmp.ODWljpQfkE_._Packages -
    # open (13: Permission denied)
          && apt-get -o Acquire::GzipIndexes=false update \
          ;; \
      esac \
      \
      && apt-get install --no-install-recommends --no-install-suggests -y \
                $nginxPackages \
                gettext-base \
                curl \
      && apt-get remove --purge --auto-remove -y &&
      rm -rf /var/lib/apt/lists/* /etc/apt/sources.list.d/nginx.list \
      \
    # if we have leftovers from building, let's purge
    # them (including extra, unnecessary build deps)
      && if [ -n "$tempDir" ]; then \
        apt-get purge -y --auto-remove \
        && rm -rf "$tempDir" /etc/apt/sources.list.d/temp.list; \
      fi \
    # forward request and error logs to docker log collector
      && ln -sf /dev/stdout /var/log/nginx/access.log \
      && ln -sf /dev/stderr /var/log/nginx/error.log \
    # create a docker-entrypoint.d directory
      && mkdir /docker-entrypoint.d
      COPY docker-entrypoint.sh /
```

```
COPY 10-listen-on-ipv6-by-default.sh /docker-entrypoint.d
COPY 15-local-resolvers.envsh /docker-entrypoint.d
COPY 20-envsubst-on-templates.sh /docker-entrypoint.d
COPY 30-tune-worker-processes.sh /docker-entrypoint.d
ENTRYPOINT ["/docker-entrypoint.sh"]

EXPOSE 80

STOPSIGNAL SIGQUIT

CMD ["nginx", "-g", "daemon off;"]
```

解説

インストールされたパッケージと更新を完全に制御する必要がある場合は、独自の Dockerfile を作成すると便利です。ベースイメージが信頼でき、本番環境で実行する前にチームによってテストされていることを確認できるように、イメージの独自のリポジトリを保持するのが一般的です。

関連項目

GitHubの公式のNGINX Dockerリポジトリ

11.5 NGINX Plusコンテナイメージの構築

問題

コンテナ化された環境でNGINX Plusを実行するために、NGINX Plus Dockerイメージを構築したいです。

解決法

F5は、Dockerfileを介してNGINX Plusをインストールするためにブログを保守して最新状態に更新しています。これらのDockerfileに基づき独自のDockerfileを構築できます。

ブログにあるDockerfileを使って、NGINX Plusのコンテナイメージを構築してください。手順は、レシピ11.4のNGINXオープンソース用のDockerfileと同じですが、NGINX Plusのリポジトリ証明書（それぞれ*nginx-repo.crt*と*nginx-repo.key*という名前）をダウンロードして、このDockerfileを配置するディレクトリに置いておく必要があります。これだけで、このDockerfileにより、NGINX Plusをインストールする残りの作業が実行されます。

`docker build`コマンドは`--no-cache`フラグを使用して、構築するたびに、NGINX Plusパッケージが NGINX Plusリポジトリから新しく取得され、更新するようにします。NGINX Plus で以前のビルドと同じバージョンを使用することが許容される場合は、`--no-cache`フラグを省略できます。この例では、新しいコンテナイメージは`nginxplus`というタグがついています：

```
$ docker build --no-cache -t nginx-plus .
```

解説

NGINX Plus用に独自のコンテナイメージを作成することで、NGINX Plusコンテナを適切な方法で構成し、任意のDocker環境にドロップできます。これにより、NGINX Plus のすべてのパワーと高度な機能がコンテナ化された環境に提供されます。このDockerfileは、Dockerfileプロパティの COPY を使用して構成に追加します。構成をローカルディレクトリに追加する必要があります。

関連項目

ブログ（英語）：“Deploying NGINX and NGINX Plus with Docker”

11.6 NGINXでの環境変数の使用

問題

異なる環境で同じコンテナイメージを使用するには、NGINX構成内で環境変数を使用したいです。

解決法

`nginx:stable-perl`のようなNGINX Perlモジュールを使用して構築された公式のNGINXコンテナイメージのいずれかを使用します。

`Load_module`ディレクティブを使用して、`nginx_http_perl_module`を有効にします。デフォルトでは、NGINXは、親プロセスから継承したTZ変数以外の環境変数をすべて削除します。`env`ディレクティブを使うことで変数を保持できますが、これらの変数は構成には公開されません。この例では、NGINXプロセス内で`APP_DNS`という環境変数を保持し、Perlモジュールの`perl_set`ディレクティブを使用して、構成で使用できる環境変数を設定します。NGINX構成に公開される環境変数の名前は`$upstream_app`です。この`$upstream_app`変数は、リクエストのプロキシ先となるドメインとして使用されます：

```
load_module /modules/
ngx_http_perl_module.so;
env APP_DNS;
# ...
http {
  perl_set $upstream_app 'sub { return $ENV{"APP_DNS"}; }';
  server {
    # ...
    location / {
      proxy_pass https://$upstream_app;
    }
  }
}
```

解説

Dockerを使用する場合の一般的な方法は、環境変数を利用してコンテナの動作方法を変更することです。NGINX構成で環境変数を使用して、NGINX Dockerfileを複数の異なる環境で使用できるようにすることができます。

11.7 NGINXのNGINX Ingress Controller

問題

アプリケーションをKubernetesにデプロイしようとしています。Ingressコントローラーを利用したいです。

解決法

Ingressコントローラーイメージにアクセスできることを確認してください。NGINXオープンソースには、Docker Hubからの*nginx/nginx-ingress*イメージを使用できます。NGINX Plusでは、F5のコンテナリポジトリからイメージを取得するか、独自のイメージを構築してプライベートコンテナイメージレジストリにホストできます。独自のNGINX Plus Kubernetes Ingressコントローラを構築し、プッシュする方法に関する指示はNGINXドキュメンテーションを参照してください。

GitHubのkubernetes-ingressリポジトリのKubernetes Ingress Controller Deploymentsフォルダを訪問してください。後続のコマンドはリポジトリのローカルコピーのディレクトリ内から実行されます。

ingressコントローラーの名前空間とサービスアカウントを作成します。両方とも同じ名前が付けられています：`nginx-ingress`:

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

オプションで、NGINX構成をカスタマイズするためのConfigMapを作成できます（提供されているConfigMapは空白ですが、ConfigMapのカスタマイズとアノテーションについてはNGINXドキュメンテーションで詳しく読むことができます）：

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

クラスタでロールベースのアクセスコントロール（RBAC）が有効になっている場合、クラスタロールを作成して、サービスアカウントにバインドします。この手順を実行するにはクラスタ管理者でなければなりません。：

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

Ingressコントローラーをデプロイします。このリポジトリでは、DeploymentとDaemonSetの2つのデプロイメント例が利用可能になっています。Ingressコントローラーのレプリカの数を動的に変更する場合は、Deploymentを使用します。すべてのノードまたはノードのサブセットにIngressコントローラーをデプロイする場合には、DaemonSetを使用します。

NGINX Plus Deploymentマニフェストを使用する場合は、YAMLファイルを変更し、独自のレジストリとイメージを指定する必要があります。

NGINX Deploymentの場合：

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

NGINX Plus Deploymentの場合：

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

NGINX DaemonSet:の場合：

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

NGINX DaemonSet:の場合：

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

Ingressコントローラーが稼働していることを確認します：

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

DaemonSetを作成した場合には、Ingressコントローラのポート`80`と`443`はコンテナが稼働しているノードの同じポートにマップされます。 Ingressコントローラーにアクセスするには、これらのポートと、Ingressコントローラーが実行されているノードのIPアドレスを使用します。Deploymentをデプロイした場合、次の手順に進みます。

Deploymentメソッドの場合、Ingressコントローラーポッドにアクセスするための2つのオプションがあります。Ingressコントローラーポッドにマップする NodePortをランダムに割り当てるようにKubernetesに指示できます。これは、タイプ`NodePort`のサービスです。もう1つのオプションはタイプ`LoadBalancer`のサービスを作成することです。タイプLoadBalancerのサービスを作成する場合、Kubernetesは、Amazon Web Services（AWS）、Microsoft Azure、Google Cloud Computeなどの特定のクラウドプラットフォーム用のロードバランサーを構築します。

タイプ`NodePort`のサービスを作成するには、以下を使用します：

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

ポッドのために開かれたポートを静的に構成するには、YAMLを変更して、属性`nodePort : {port}`を開かれた各ポートの構成に追加します。

Google Cloud ComputeまたはAzureのためにタイプ`LoadBalancer`のサービスを作成する場合には、このコードを使用します：

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

AWSのためにタイプLoadBalancerのサービスを作成する：

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

AWSで、KubernetesはPROXYプロトコルを有効にしてTCPモードでクラシックELBを作成します。これを実行するには、ファイル*common/nginx-config.yaml*を参照して、前述のConfig Mapに以下を追加します：

```
proxy-protocol: "True"
```

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

その後、コンフィグマップを更新します：

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

これで、NodePortを使用するか、ポッドに代わって作成されたロードバランサーにリクエストを送信することで、ポッドにアドレスできます。

解説

執筆時点では、Kubernetesはコンテナオーケストレーションと管理のリーダー的プラットフォームです。Ingressコントローラーは、トラフィックをアプリケーションのその他の部分にルーティングするエッジポッドです。NGINXはこの役割に完全に適合し、注釈を使用した構成をシンプルにします。NGINX Ingressプロジェクトは、Docker Hubイメージからすぐに使用できるNGINX Open Source Ingressコントローラーを提供します。NGINX Plusでは、F5コンテナリポジトリを使用していくつかのステップを介してリポジトリ証明書と鍵を追加できます。NGINX Ingressコントローラーを使用してKubernetesクラスタを有効にすると、NGINXと同じ機能がすべて提供されますが、トラフィックをルーティングするためのKubernetesネットワークとDNSの機能が追加されます。

第12章
高可用性展開モード

12.0 はじめに

フォールトトレラントアーキテクチャは、システムを同一の独立したスタックに分離します。NGINXなどのロードバランサーは負荷を分散し、プロビジョニングされたものが確実に利用されるようにします。高可用性のコアコンセプトは、複数のアクティブノードでの負荷分散またはアクティブ-パッシブフェイルオーバーです。高可用性アプリケーションには単一障害点がありません。ロードバランサー自体を含め、すべてのコンポーネントはこれらのコンセプトの1つを使用する必要があると言えます。私たちにとっては、それがNGINXです。NGINXは、複数のアクティブまたはアクティブ-パッシブフェイルオーバーのいずれかの構成で機能するように設計されています。本章では、複数のNGINXサーバーを実行して、負荷分散層で高可用性を確保する方法について詳しく説明します。

12.1 NGINX Plus HAモード

問題

オンプレミス導入で高可用性（HA）ロードバランシングソリューションが必要です。

解決法

NGINX Plusリポジトリから`nginx-ha-keepalived`パッケージを複数のシステムにインストールして、keepalivedでNGINX PlusのHAモードを使用します：

```
$ sudo apt update
$ sudo apt install -y nginx-ha-keepalived
```

`nginx-ha-setup`スクリプトを使用して、各システムでkeepalived構成ファイルをブートストラップし、プロンプトに従います：

```
$ sudo nginx-ha-setup
```

`nginx-ha-setup`コマンドによって生成された構成ファイルを確認します：

```
$ sudo cat /etc/keepalived/keepalived.conf

global_defs {
vrrp_version 3
}

vrrp_script chk_manual_failover {
  script "/usr/lib/keepalived/nginx-ha-manual-failover"
  interval 10
  weight 50
}

vrrp_script chk_nginx_service {
  script "/usr/lib/keepalived/nginx-ha-check"
  interval 3
  weight 50
}

vrrp_instance VI_1 {
  interface eth0
  priority 101
  virtual_router_id 51
  advert_int 1 accept
  garp_master_refresh 5
  garp_master_refresh_repeat 1
  unicast_src_ip 172.17.0.2/16
  unicast_peer {
    172.17.0.4
  }
  virtual_ipaddress {
    172.17.0.3
  }
  track_script {
    chk_nginx_service
    chk_manual_failover
  }
  notify "/usr/lib/keepalived/nginx-ha-notify"
}
```

構成されたヘルスチェックスクリプトをテストして、ノードが正常であることを確認します：

```
$ sudo/usr/lib/keepalived/nginz-ha-check

nginx is running.
```

セカンダリノードで、VRRP拡張統計とデータをファイルシステムにダンプし、出力を確認します：

```
$ sudo service keepalived dump

Dumping VRRP stats (/tmp/keepalived.stats)
and data (/tmp/keepalived.data)

$ sudo cat /tmp/keepalived.stats

VRRP Instance: VI_1
  Advertisements:
    Received: 1985
    Sent: 0
  Became master: 0
  Released master: 0
  Packet Errors:
    Length: 0
    TTL: 0
    Invalid Type: 0
    Advertisement Interval: 0
    Address List: 0
  Authentication Errors:
    Invalid Type: 0
    Type Mismatch: 0
    Failure: 0
  Priority Zero:
    Received: 0
    Sent: 0
```

プライマリノードの状態を強制的に変更します：

```
$ sudo service keepalived stop

Stopping keepalived: keepalived.
```

再び、セカンダリノードで、VRRP拡張統計とデータをファイルシステムにダンプし、出力を確認します：

```
$ sudo service keepalived dump

Dumping VRRP stats (/tmp/keepalived.stats)
and data (/tmp/keepalived.data)

$ sudo cat /tmp/keepalived.stats

VRRP Instance: VI_1
  Advertisements:
    Received: 1993
    Sent: 278
  Became master: 1
  Released master: 0
```

```
  Packet Errors:
    Length: 0
    TTL: 0
    Invalid Type: 0
    Advertisement Interval: 0
    Address List: 0
  Authentication Errors:
    Invalid Type: 0
    Type Mismatch: 0
    Failure: 0
  Priority Zero:
    Received: 0
    Sent: 0
```

統計ファイルで、当初セカンダリだったノードが少なくとも1回はプライマリノードになっていることが報告されていることに注意してください。

解説

`nginx-ha-keepalived`パッケージはkeepalivedに基づき、クライアントに表示される仮想 IPアドレスを管理します。このソリューションは、標準的なオペレーティングシステムコールによってIPアドレスを制御できる環境で動作するように設計されているので、IPアドレスがクラウドインフラストラクチャとのインターフェースによって制御されるクラウド環境では動作しないことがよくあります。

Keepalivedは、仮想ルータ冗長プロトコル（VRRP）を利用するプロセスであり、ハートビートと呼ばれる小さなメッセージをバックアップサーバーに送信します。バックアップサーバーが3回連続してハートビートを受信しない場合、バックアップサーバーはフェイルオーバーを開始し、仮想IPアドレスを自分に移動してプライマリになります。`nginx-ha-keepalived`のフェイルオーバー機能は、カスタムの障害状況を識別するように構成できます。`nginx-ha-setup`スクリプトは基本構成を提供することだけを目的としています。keepalivedをさらにカスタマイズするには、その構成ファイルを編集してサービスを再起動してください。

関連項目

Keepalived ドキュメンテーション

12.2 DNSでのロードバランシングロードバランサー

問題

2つ以上のNGINXサーバー間で負荷を分散する環境にしたいです。

解決法

複数のIPアドレスをDNS A レコードに追加して、DNSを使用してNGINXサーバー間にラウンドロビンを適用します。

解説

複数のロードバランサーを稼働している場合は、DNSを介して負荷を分散できます。Aレコードを使用すると、複数のIPアドレスを単一のFQDNの下にリストできます。DNSは、リストされているすべてのIP間で自動的にラウンドロビンを実行します。DNSは、重み付きレコードを使用した重み付きラウンドロビンも提供します。これは、第2章で説明されているNGINXの重み付きラウンドロビンと同様に機能します。これらのテクニックは非常に優れています。しかし、問題点は、NGINXサーバーで障害が発生した場合の、レコードの削除です。Amazon Route 53やDyn DNSなどのヘルスチェックを行い、自社のDNS製品にフェイルオーバーを行うDNSプロバイダーがあり、こうした問題を回避できます。DNSを使用してNGINXのロードバランスを行っている場合、NGINXサーバーに削除のマークが付けられている時は、NGINXがアップストリームサーバーを削除する場合に行うのと同じプロトコルに従うのが最善です。まず、DNSレコードから IPを削除して、新しい接続の送信を停止します。次に、サービスを停止またはシャットダウンする前に、接続のドレインを許可します。

12.3 EC2でのロードバランシング

問題

AWSでNGINXを使用しており、NGINX Plus HAでAmazonで利用できるIPアドレスの付け替えはサポートしていますか。

解決法

NGINXサーバーのAuto Scalingグループを構成し、そのAuto Scalingグループをターゲットグループにリンクします。そして、そのターゲットグループをNLBにアタッチすることで、NGINXをAWS NLBの背後に配置できます（レシピ10.5参照）。その他に、AWSコンソール、コマンドラインインターフェース、APIを使用して、手動でNGINXサーバーをターゲットグループに配置することもできます。

解説

keepalivedに基づくNGINX PlusのHAソリューションは、AWSでは機能しません。EC2 IPアドレスの動作が異なることを理由に、AWSはフローティング仮想IPアドレスをサポートして

いないためです。しかし、だからと言って、NGINXがAWSクラウドでHAになれないという意味ではなく、実際には、その逆です。AWS NLBは、*アベイラビリティーゾーン*と呼ばれる物理的に分離された複数のデータセンター間でネイティブにロードバランスして、アクティブなヘルスチェックとDNS CNAMEエンドポイントを提供するAmazonの製品です。AWSのHA NGINXの一般的な解決策は、NLBの背後にNGINXレイヤーを配置することです。NGINXサーバーは、必要に応じてターゲットグループに自動的に追加および削除できます。NLBはNGINXの代替品ではありません。複数のロードバランシング方法、レート制限、キャッシング、レイヤー7ルーティングなど、NGINXは提供するものの、NLBが提供していない機能が多数あります。AWS ALBは、URIパスとホストヘッダーに基づいてレイヤー7ロードバランシングを実行しますが、WAFキャッシング、帯域幅制限などのNGINXが提供する機能をALB自体では提供しません。NLBがニーズに合わない場合は、他にも多くのオプションがあります。1つのオプションがDNSソリューションです：AWSのRoute 53はヘルスチェックとDNSフェイルオーバーを提供します。

12.4 NGINX Plus構成の同期

問題

HA NGINX Plus層を運用しており、サーバー間で構成を同期したいです。

解決法

NGINX Plusの同期機能を使用します。この機能を構成するには、以下の手順に従います。

`nginx-sync`パッケージをNGINX Plusパッケージリポジトリからインストールします

YUMパッケージマネージャーでNGINX Plusをインストール：

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

APTパッケージマネージャーでNGINX Plusをインストール：

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

プライマリマシンにルートとして同期先マシンへのSSHアクセスを許可します。

ルートにSSH認証キーペアを生成し、公開キーを取得します：

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

プライマリノードのIPアドレスを取得します：

```
$ 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
```

`ip addr`コマンドはマシンのインターフェースに関する情報を取得します。ループバックインターフェースは無視し。`inet`の後にあるプライマリインターフェースのIPアドレスを探します。この例では、IPアドレスは`192.168.1.2`です。

公開鍵を各同期先ノードのルートユーザーの*authorized_keys*ファイルに配布し、プライマリIPアドレスからのみ承認するように指定します：

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

次の行を*/etc/ssh/sshd_config*に追加し、すべてのノードで`sshd`をリロードします：

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

プライマリノードのルートユーザーがパスワードなしで各同期先ノードに`ssh`接続できることを確認します：

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

構成ファイル*/etc/nginx-sync.conf*をプライマリマシンに以下の構成で作成します：

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

この構成例は、この機能の3つの一般的な構成パラメータを示しています：`NODES`、`CONFIGPATHS`、`EXCLUDE`です。`NODES`パラメータは、スペースで区切られたホスト名または IPアドレスの文字列に設定されます。これらは、プライマリが構成変更をプッシュする同期先ノードです。`CONFIGPATHS`パラメータは、同期する必要のあるファイルまたはディレクトリを示します。最後に、`EXCLUDE`パラメータを使用して、構成ファイルを同期から除外できます。この例では、プライマリはメインのNGINX構成ファイルの構成変更をプッシュし、ディレクトリ*/etc/nginx/nginx.conf* および*/etc/nginx/conf.d*を`node2.example. comnode3.example.com`および`node4.example.com`という名前の同期先ノードに含めます。同期プロセスで*default.conf*という名前のファイルが見つかった場合、そのファイルは`EXCLUDE`として構成されているため、同期先にプッシュされません。

NGINXバイナリ、RSYNCバイナリ、SSHバイナリ、diffバイナリ、ロックファイルのロケーションおよびバックアップディレクトリの場所を構成するための高度な構成パラメータがあります。テンプレートで指定されたファイルに`sed`を利用するパラメータもあります。高度なパラメー

タに関する詳細情報は、「クラスタでのNGINX構成の共有」でNGINXドキュメンテーションを参照してください。

構成のテスト：

```
$ 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
```

解説

このNGINX Plus独自の機能を使用すると、プライマリノードのみを更新し、構成を他のすべての同期先ノードに同期することで、HA構成内の複数のNGINX Plusサーバーを管理できます。構成の同期を自動化することにより、構成を転送する際の誤りのリスクを制限します。*. nginx-sync.sh*アプリケーションは正常でない構成を同期先に送信するのを防止するセーフガードを提供します。これには、プライマリでの構成のテスト、同期先での構成のバックアップの作成、およびリロードする前の同期先の構成の検証が含まれます。構成管理ツールまたはDockerを使用して構成を同期することをお勧めしますが、この方法で環境を管理できる状況にない場合は、NGINX Plus構成同期機能が役立ちます。

12.5 NGINX PlusおよびZone Syncを使用した状況共有

問題

NGINX Plusの高可用性サーバー間で共有メモリーゾーンを同期したいです。

解決法

ゾーン同期を構成してから、NGINX Plus共有メモリーゾーンを構成する際にsyncパラメータを使用します。

```
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;
        }
    }
}
```

解説

zone_syncモジュールはNGINX Plusだけで利用できる機能で、NGINX Plusをより良いクラスタにするものです。構成に示されているように、zone_syncとして構成されたstreamサーバーをセットアップする必要があります。例で、ポート9000をリッスンしているサーバーがこれです。NGINX Plusはその他のサーバーとzone_sync_serverディレクティブによって通信します。このディレクティブを、動的クラスタの複数のIPアドレスに解決されるドメイン名に設定するか、一連のzone_sync_serverディレクティブを静的に定義して、単一障害点を回避できます。ゾーン同期サーバーへのアクセスを制限する必要があります。このモジュールには、マシン認証用の特定のSSL/TLSディレクティブがあります。

NGINX Plusをクラスタに構成する利点は、共有メモリーゾーンを同期して、レート制限、スティッキーラーンセッション、およびキーバリューストアを実現できることです。提示の例は、sticky learnディレクティブの最後に付けられたsyncパラメータを示します。この例では、ユーザーはsessionという名前のcookieに基づいてアップストリームサーバーにバインドされています。zone_syncモジュールがないと、ユーザーが別のNGINX Plusサーバーに要求を送信すると、セッションが失われる可能性があります。zone_syncモジュールを使用すると、すべてのNGINX Plusサーバーがセッションと、セッションがバインドされているアップストリームサーバーを認識します。

第13章
高度なアクティビティモニタリング

13.0 はじめに

アプリケーションが最適なパフォーマンスと精度で実行されていることを確認するには、そのアクティビティに関する監視メトリクスのインサイトが必要です。NGINXは、スタブステータスや高度な監視ダッシュボード、NGINX PlusのJSONフィードなど、さまざまな監視オプションを提供しています。NGINX Plusアクティビティモニタリングは、リクエスト、アップストリームサーバープール、キャッシング、ヘルスなどに関するインサイトを提供します。OpenTelemetryを使用することで、さらにアプリケーションに統合された可観測性も利用できます。本章では、NGINXの機能と可能性について詳しく説明します。

13.1 NGINXスタブステータスの有効化

問題

NGINXのベーシックモニタリングを有効にする必要があります。

解決法

NGINX HTTPサーバーのロケーションブロックでstub_statusモジュールを有効にします：

```
location /stub_status {
  stub_status;
  allow 127.0.0.1;
  deny all;
  # Set IP restrictions as appropriate
}
```

ステータスにリクエストを作成して、構成を確認します。

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

解説

stub_statusモジュールはNGINX OSSサーバーの一部のベーシックモニタリングを有効にします。返される情報は、アクティブな接続の数、受け入れられた接続の合計数、対応された接続、および処理されたリクエストに関するインサイトを提供します。読み取り、書き込み、または待機状態にある現在の接続数も表示されます提供される情報はグローバルであり、stub_statusディレクティブが定義されている親サーバーに固有のものではありません。つまり、ステータスを保護対象のサーバーにホストできるということです。セキュリティ上の理由から、ローカルトラフィックを除く監視機能へのすべてのアクセスをブロックしました。このモジュールは、ログやその他の場所で使用するための埋め込み変数としてアクティブな接続数を提供します。その変数とは、$connections_active、$connections_reading、$connections_writing、$connections_waitingです。

13.2 NGINX Plus監視ダッシュボードの有効化

問題

NGINX Plusサーバーを流れるトラフィックに関する詳細なメトリクスが必要です。

解決法

リアルタイムアクティビティ監視ダッシュボードを活用します：

```
server {
  # ...
  location /api {
    api [write=on];
    # Directives limiting access to the API
    # See chapter 7
  }

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

NGINX Plus構成はNGINX Plusステータス監視ダッシュボードを提供します。この構成は、APIとステータスダッシュボードを提供するHTTPサーバーをセットアップします。ダッシュボードは、*/usr/share/nginx/html*ディレクトリから静的コンテンツとして提供されます。ダッシュボードは、ステータスをリアルタイムで取得して表示するために、*/api/*にあるAPIにリクエストを送信します。

解説

NGINX Plusは高度なステータス監視ダッシュボードを提供します。このステータスダッシュボードは、アクティブな接続の数、稼働時間、アップストリームサーバープール情報など、NGINXシステムの詳細なステータスを提供します。コンソールに関する情報は図13-1を参照してください。

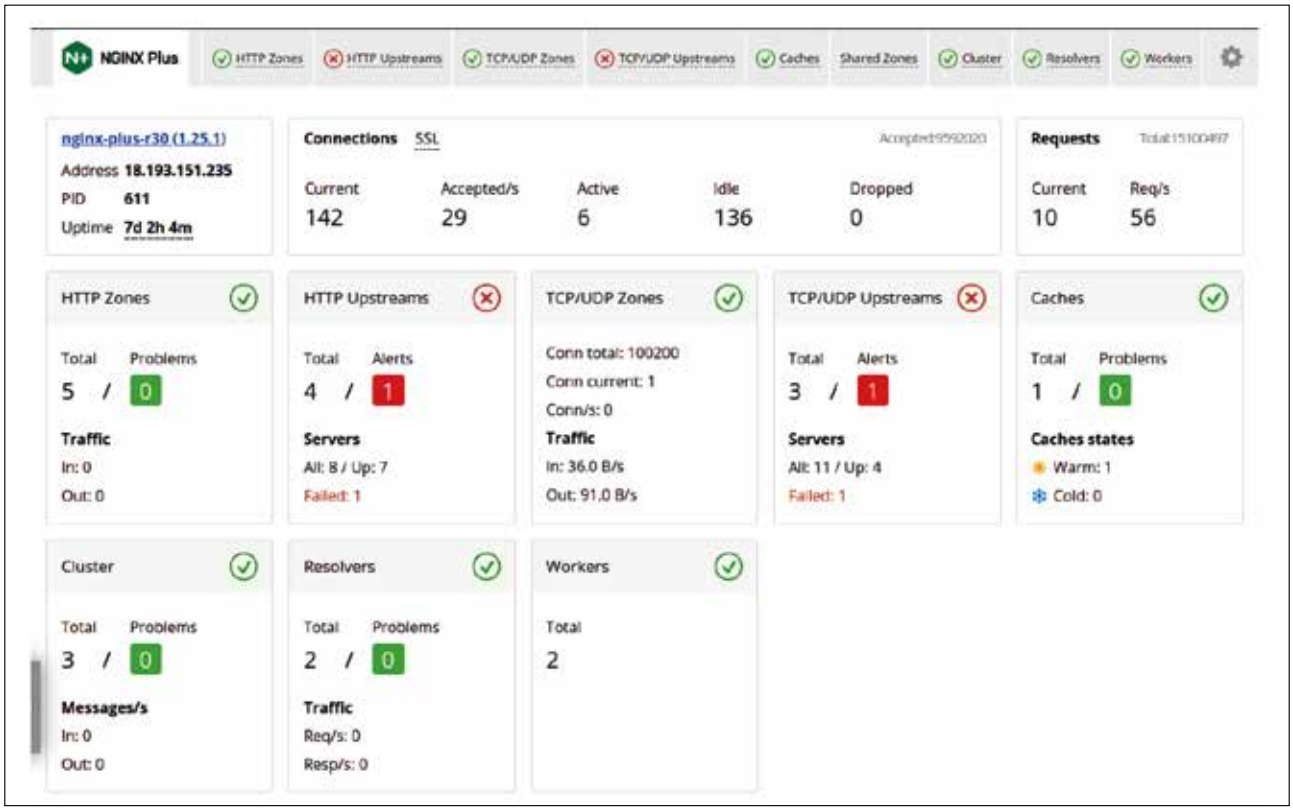


図13-1　NGINX Plusステータスダッシュボード

ステータスダッシュボードのランディングページには、システム全体の概要が表示されます。[HTTPゾーン]タブをクリックすると、NGINX構成で構成されたすべてのHTTPサーバーの詳細が一覧表示され、1XXから5XXまでの応答数、全体の合計、1秒あたりのリクエスト数、現在のトラフィックスループットが表示されます(図13-2)。[HTTPアップストリーム]タブには、アップストリームサーバーのステータスの詳細が表示されます。サーバーが障害状態にあった場合、サーバーが処理したリクエストの数、ステータスコードによって処理された応答の合計数、およびヘルスチェックの合格または不合格の数などの他の統計情報です。[TCP/UDPゾーン]タブには、TCPまたはUDPストリームを流れるトラフィックの量と接続数の詳細が表示されます。[TCP/UDPゾーン]タブには、TCPまたはUDPストリームを流れるトラフィックの量と接続数の詳細が表示されます。[TCP/UDPアップストリーム]タブには、TCP/UDPアップストリームプール内の各アップストリームサーバーがサービスを提供している量、ヘルスチェックの合

格と不合格の詳細および応答時間に関する情報が表示されます。[キャッシュ]タブには、キャッシュに使用されているスペースの量に関する情報が表示されます。提供、書き込み、およびバイパスされたトラフィックの量。ヒット率も同様です。NGINXステータスダッシュボードは、アプリケーションの中心とトラフィックフローを監視する上で役立ちます。

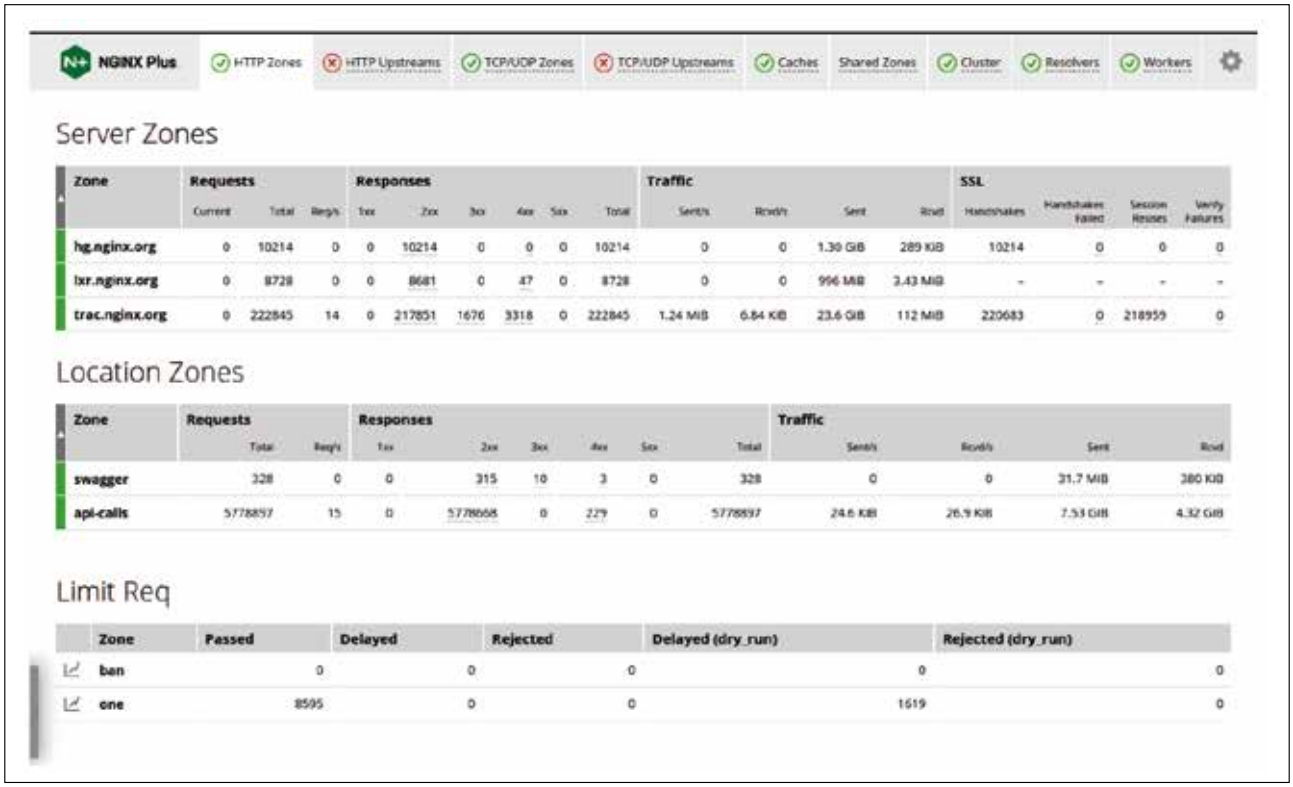


図13-2　NGINX Plusステータスダッシュボード[HTTPゾーン]ページ

関連項目

Demo: NGINX Plus ステータスダッシュボード

13.3 NGINX Plus APIを使用したメトリクスの収集

問題

NGINX Plusステータスダッシュボードによって提供される詳細メトリクスへのAPIアクセスが必要です。

解決法

RESTfulAPIを利用してメトリクスを収集します。この例では、出力を`json_pp`にパイプして、読みやすくします：

```
$ curl "https://demo.nginx.com/api/9/" | json_jq
[
  "nginx",
  "processes",
  "connections",
  "slabs",
  "http",
  "stream",
  "resolvers",
  "ssl",
  "workers"
]
```

curl呼び出しはAPIのトップレベルをリクエストします。それは、APIのその他の部分も表示します。

NGINX Plusサーバーに関する情報は*/api/{version}/nginxURI*を使用します：

```
$ curl "https://demo.nginx.com/api/9/nginx" | json_jq
{
  "address" : "10.3.0.7",
  "build" : "nginx-plus-r30",
  "generation" : 1,
  "load_timestamp" : "2023-08-15T11:38:35.603Z",
  "pid" : 622,
  "ppid" : 621,
  "timestamp" : "2023-08-22T20:49:21.717Z",
  "version" : "1.25.1"
}
```

APIにより返される情報を制限するには、引数を使用します：

```
$ curl "https://demo.nginx.com/api/9/nginx?fields=version,build" \
  | json_jq
{
  "build" : "nginx-plus-r30",
  "version" : "1.25.1"
}
```

*/api/{version}/connections*からの接続統計をURIから要求できます：

```
$ curl "https://demo.nginx.com/api/9/connections" | json_jq
{
  "accepted" : 9884385,
  "active" : 4,
  "dropped" : 0,
  "idle" : 127
}
```

*/api/{version}/http/requests*からのリクエスト統計はURIから収集できます：

```
$ curl "https://demo.nginx.com/api/9/http/requests" | json_jq
{
  "total" : 52107833,
  "current" : 2
}
```

*/api/{ver sion}/http/server_zones/{httpServerZoneName}*URIを使用して特定のサーバーゾーンに関する統計を取得できます：

```
$ curl "https://demo.nginx.com/api/9/http/server_zones/hg.nginx.org" \
  | json_jq
{
  "discarded" : 0,
  "processing" : 0,
  "received" : 308357,
  "requests" : 10633,
  "responses" : {
    "1xx" : 0,
    "2xx" : 10633,
    "3xx" : 0,
    "4xx" : 0,
    "5xx" : 0,
    "codes" : {
      "200" : 10633
    },
    "total" : 10633
  },
  "sent" : 1327232700,
  "ssl" : {
    "handshake_timeout" : 0,
    "handshakes" : 10633,
    "handshakes_failed" : 0,
    "no_common_cipher" : 0,
    "no_common_protocol" : 0,
    "peer_rejected_cert" : 0,
    "session_reuses" : 0,
    "verify_failures" : {
      "expired_cert" : 0,
      "no_cert" : 0,
      "other" : 0,
      "revoked_cert" : 0
    }
  }
}
```

APIは、ダッシュボードに表示される任意のデータを返すことができます。これには深度があり、論理的パターンがあります。このレシピの最後にリソースへのリンクが記載されています。

解説

NGINX Plus APIは、NGINX Plusサーバーの多くの部分に関する統計を返すことができます。NGINX Plusサーバーに関する情報、そのプロセス、接続、スラブに関する情報を収集できます。サーバー、アップストリーム、アップストリームサーバー、キーバリューストアなど、NGINX内で実行されているhttpとstreamサーバーに関する情報、およびHTTPキャッシュゾーンに関する情報と統計も確認できます。これにより、ユーザーまたはサードパーティのメトリクスアグリゲーターに、NGINX Plusサーバーのパフォーマンスの詳細なビューが提供されます。

関連項目

NGINX HTTP APIモジュールドキュメンテーション

NGINX API REST UI

「Metrics APIの使用」チュートリアル

13.4 OpenTelemetry for NGINX

問題

OpenTelemetry（OTel）をサポートするトレースコレクターがあり、NGINXと統合する必要があります。

解決法

NGINXのパッケージリポジトリからNGINX OTelをインストールします。

YUMパッケージマネージャーでNGINX Plusをインストール：

```
$ sudo yum install -y nginx-plus-module-otel
```

APTパッケージマネージャーでNGINX Plusをインストール：

```
$ sudo apt install -y nginx-plus-module-otel
```

YUMパッケージマネージャーでNGINXオープンソースをインストール：

```
$ sudo yum install -y nginx-module-otel
```

APTパッケージマネージャーでNGINXオープンソースをインストール：

```
$ sudo apt install -y nginx-module-otel
```

NGINX OTel動的モジュールをロードし、OTel対応のレシーバーを構成する：

```
load_module modules/ngx_otel_module.so;
# ...
http {

  otel_exporter {
    endpoint localhost:4317;
```

```
    }

    server {
      listen 127.0.0.1:8080;

      location / { otel_trace on;
        otel_trace_context inject;

        proxy_pass http://backend;
      }
    }
  }
```

NGINXは、着信リクエストからトレースコンテキストを継承し、スパンを記録して、otel_trace_conextディレクティブをpropagateに設定することでトレースをバックエンドサーバーに伝播します：

```
  server {
    location / {
      otel_trace on;
      otel_trace_context propagate;
      proxy_pass http://backend;
    }
  }
```

着信リクエストからトレースコンテキストを継承し、split_clientsと組み合わせてotel_traceを有効にするだけで、トラフィックの10%のスパンだけを記録できます：

```
  # trace 10% of requests
  split_clients "$otel_trace_id" $ratio_sampler {
    10% on;
    * off;
  }

  server {
    location / {
      otel_trace $ratio_sampler;
      otel_trace_context propagate;

      proxy_pass http://backend;
    }
  }
```

問題をデバッグするときなど、すべてのリクエストに対してトレースが有効になるようにしたい場合があります。NGINXのmap機能を使用すると、OR論理ゲートを作成できます。先ほどの例を変更した以下の例では、map機能を使って変数にonかoffの値を割り当てます。mapは、ENABLE_OTEL_TRACEというHTTPヘッダーに基づきます。別のmapを使用して変数を生成し、リクエストが10% split_clientsの場合、またはENABLE_OTEL_TRACEヘッダーが存在する場合、その値をonにします：

```
# trace 10% of requests
split_clients "$otel_trace_id" $ratio_sampler {
  10% on;
  * off;
}

# Always trace when ENABLE_OTEL_TRACE header has a value
map "$http_enable_otel_trace" $has_trace_header {
  default on;
  '' off;
}

# Or statement to turn on otel_trace if either of the
# above cases enable it
map "$ratio_sampler:$has_trace_header" $request_otel {
  off:off  off;
  on:on    on;
  on:off   on;
  off:on   on;
}

server {
  location / {
    otel_trace $request_otel;
    otel_trace_context propagate;

    proxy_pass http://backend;
  }
}
```

NGINX Plusを使用している場合は、NGINX Plus APIとキーバリューストアを利用することで、特定のセッションの100%トレースを有効/無効にできます：

```
# KV store to enable dynamic 100% tracing
keyval "otel.trace" $trace_switch zone=otel_kv;

# trace 10% of requests
split_clients "$otel_trace_id" $ratio_sampler {
  10%     on;
  *       off;
}

# Or statement to turn on otel_trace if any of the
# above cases enable it
map "$trace_switch:$ratio_sampler" $request_otel {
  off:off  off;
  on:on    on;
  on:off   on;
```

```
    off:on   on;
}

server {
    location / {
        otel_trace $request_otel;
        otel_trace_context propagate;

        proxy_pass http://backend;
    }
    location /api {
        api write=on;
    }
}
```

解説

OpenTelemetryは、アプリケーション計測に使用されるSDK、APIおよびツールのコレクションであり、テレメトリデータを生成および収集します。このデータは、JaegerやPrometheusのようなシステムにストリーミングされます。これにより、正確な可観測性が提供され、エンジニアはアプリケーションスタックを介してコールをトレースできます。NGINX OTelモジュールは、リクエストに関するデータを収集し、リクエストに`tracestate`および`traceparent`ヘッダーを設定して、このデータをOpenTelemetryレシーバーに送ることで、これらのテレメトリシステムと統合します。

このセクションでは、NGINX `ngx_otel_module`は、動的にロードされ、テレメトリデータを送信するレシーバーを提供しています。`otel_trace_context`ディレクティブは、`traceparent`および`tracestate`ヘッダーを抽出、注入、伝播または無視するように構成できますが、ここでは、伝播するように構成されています。`propagate`パラメータは、`extract`メソッドと`inject`メソッドを組み合わせたものです。`extract`メソッドはHTTPヘッダーから`tracestate`と`traceparent`を取得し、`inject`メソッドは新しい値を作成してアップストリームリクエストに設定します。`propagate`を使用する場合、新しいIDは、HTTPヘッダーがまだ設定されていない場合にのみ作成されます。

多忙なシステムでは、すべてのリクエストのトレースデータを記録すると少々多すぎるかもしれません。この例では、`split_clients`モジュールを使ってリクエストの10%のトレースデータだけを送る方法を説明しました。この例をさらに改良した例では、開発者は、リクエストヘッダーを設定してトレースを有効にすることで、リクエストを確実にトレースできるようになりました。これらのオプションは、`map`モジュールと結合され、OR論理ゲートを作成し、両方のケースでoffが返された場合にのみトレースが`off`に設定されます。

関連項目

NGINX Plus OTelモジュールドキュメンテーション

「NGINX Plus R29の発表」（OpenTelemetryサポート）

OpenTelemetryドキュメンテーション

GitHubのNGINX OpenTelemetryプロジェクト

Split Clientsモジュールドキュメンテーション

NGINX Plus APIモジュールドキュメンテーション

NGINX Plusキーバリューモジュールドキュメンテーション

NGINXオープンソースOpenTelemetryモジュール

13.5 Prometheus Exporterモジュール

問題

Prometheusモニタリングを使用した環境にNGINXを展開しようとしているため、NGINXと連携したいです。

解決法

NGINX Prometheus Exporterを使用して、NGINX統計を収集し、Prometheusに送信します。

NGINX PrometheusExporterモジュールはGoLangで記述されており、バイナリとしてGitHubで配布されています。プリビルトのコンテナイメージはDocker Hubで見つけることができます。

デフォルトでは、エクスポータはNGINXを対象に開始され、`stub_status`情報のみを収集します。NGINXオープンソースのためにエクスポータを実行するには、スタブステータスが有効になっていることを確認してください（有効になっていない場合には、その方法についての詳細情報はレシピ13.1にあります）。その後、Dockerコマンドを使用します：

```
$ docker run -p 9113:9113 nginx/nginx-prometheus-exporter:0.8.0 \
    -nginx.scrape-uri http://{nginxEndpoint}:8080/stub_status
```

NGINX Plusでエクスポータを使用するには、NGINX Plus APIからより多くのデータを収集できるため、フラグを使用してエクスポータのコンテキストを切り替える必要があります。NGINX Plus APIを有効にする方法についての詳細はレシピ13.2をご覧ください。次のDockerコマンドを使用して、NGINX Plus環境のエクスポータを実行します：

```
$ docker run -p 9113:9113 nginx/nginx-prometheus-exporter:0.8.0 \
    -nginx.plus -nginx.scrape-uri http://{nginxPlusEndpoint}:8080/api
```

解説

Prometheusは、Kubernetesエコシステムで普及している非常に一般的なメトリクス監視ソリューションです。NGINX Prometheus Exporterモジュールは非常にシンプルなコンポーネントですが、一般的な監視プラットフォームとNGINXを予め構成された形で統合することができるようになります。NGINXを使用すると、`stub_status`は大量のデータは提供しませんが、そのデータは、NGINXノードが処理している作業量に関するインサイトを提供するために重要で

す。NGINX Plus APIは、NGINX Plusサーバーに関するより多くの統計を提供できるようにします。これらはすべて、エクスポータがPrometheusに送信します。いずれの場合でも、収集された情報は貴重な監視データであり、このデータをPrometheusに送信する作業はすでに完了していますので、Prometheusに接続して、NGINX統計情報を確認することができます。

関連項目

NGINX Prometheus Exporter GitHub

NGINX stub_statusモジュールドキュメンテーション

NGINX Plus APIモジュールドキュメンテーション

NGINX Plus監視ダッシュボード

第14章

アクセスログ、エラーログ、リクエストトレースによるデバッグとトラブルシューティング

14.0 はじめに

ログは、アプリケーションを理解するための基礎です。NGINXなら、ユーザーとアプリケーションにとって意味のあるログ情報を細かく制御できます。NGINXを使用すると、アクセスログをさまざまなコンテキストのさまざまなファイルと形式に分割し、エラーログのログレベルを変更して、何が起こっているのかをより深く理解できます。集中型サーバーにログをストリーミングする機能は、Syslogログ機能を通じて提供されます。NGINXは、リクエストがシステム内を移動する際にトレースすることもできます。本章では、アクセスログとエラーログ、Syslogプロトコルを介したストリーミング、NGINXとOpenTracingによって生成されたリクエスト識別子を使用してリクエストをエンドツーエンドで追跡する方法について説明します。

14.1 アクセスログの設定

問題

アクセスログの形式を設定して埋め込み変数をリクエストログに追加する必要があります。

解決法

アクセスログの形式を設定します：

```
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"';
   # ...
}
```

このログ形式の構成はgeoproxyという名前で、NGINXログ機能を実行するために埋め込み変数を使用します。この構成は、リクエストが行われたときのサーバーの現地時間、接続を開いたIPアドレス、およびクライアントのIPを示します。NGINXはgeoip_proxyまたはrealip_headerの指示に従って理解します。GeoIPモジュールは、デフォルトではNGINXに含まれていないので、このモジュールが構成された状態でNGINXをソースから構築する必要があります。GeoIPモジュールをインストールする方法については、レシピ3.2を参照してください。

サーバーのlistenディレクティブでproxy_protocolパラメータが使用されている場合、$proxy_protocol_server_で始まる変数は、PROXYプロトコルヘッダーからサーバーに関する情報を提供します。$remote_userは、ベーシック認証によって認証されたユーザーのユーザー名を示し、その後、要求の方法とプロトコル、HTTPやHTTPSなどのスキームが続きます。サーバー名一致がログに記録され、リクエストURIと返答のステータスコードも記録されます。

ログに記録される統計には、ミリ秒単位の処理時間とクライアントに送信される本文のサイズが含まれます国、地方、都市の情報も記録されます。リクエストが別のプロキシによって転送されているかどうかを示すために、HTTPヘッダーX-Forwarded-Forが含まれます。upstreamモジュールは、いくつかの埋め込み変数を有効にします。これらの変数はアップストリームサーバーから返されたステータスとアップストリームリクエストが返されるまでにかかる時間を示します。最後に、クライアントが参照された場所と、クライアントが使用しているブラウザに関する情報をログが記録されています。

オプションのescapeパラメータは文字列でどのタイプのエスケープがなされたかを指定できます。エスケープ値はdefault、json、noneのいずれかです。noneはエスケープを無効にします。defaultエスケープでは、文字“"”、“\”および値が32未満または126を超えるその他の文字は、“\xXX”としてエスケープされます。変数値が見つからない場合には、ハイフン(“-”)が記録されます。jsonエスケープの場合、JSON文字列で許可されないすべての文字がエスケープされます。“"”や“\”は“\"”、“\\”としてエスケープされます。値が32未満の文字は“\n”、“\r”、“\t”、“\b”、“\f”、“\u00X”としてエスケープされます。

このログ構成は、次のようなログエントリをレンダリングします:

```
[25/Nov/2016:16:20:42 +0000] 10.0.1.16 192.168.0.122 Derek
GET HTTP/1.1 http www.example.com / 200 0.001 370 USA MI
"Ann Arbor" - 200 0.001 "-" "curl/7.47.0"
```

このログ形式を使用するには、パラメータとしてログファイルのパスと形式名`geoproxy`を指定して、`access_log`ディレクティブを使用します。これは、このソリューションの最初のステップで作成されたログ形式名を参照します。

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

`access_log`ディレクティブはログファイルパスと形式名をパラメータとして取ります。このディレクティブは多くのコンテキストで有効で、各コンテキストは異なるログバスおよび/またはログ形式を取ることができます。`buffer`、`flush`、`gzip`などのパラメータはログファイルにログが記録される頻度とファイルが圧縮されているかどうかを構成します。`if`ブロックと混同しないようにしてください。`access_log`ディレクティブには`if`という名前のパラメータが存在し、条件を設定します。この条件が`0`または空の文字列と評価される場合、アクセスは記録されません。

解説

NGINXのログモジュールを使用すると、さまざまなシナリオのログ形式を構成して、必要に応じて多数のログファイルに記録できます。コンテキストごとに異なるログ形式を構成して、さまざまなモジュールを使用するのが便利な場合もあるでしょう。そうしたモジュールの埋め込み変数を使用するほか、必要なすべての情報を提供する単一のキャッチオール形式を構成することもできます。そうした方法で形式の文字列を作成すれば、JSONまたはXMLでログインすることもできます。これらのログは、トラフィックパターン、クライアントの使用状況、クライアントが誰であるか、クライアントの所在地を理解するのに役立ちます。アクセスログは、アップストリームサーバーまたは特定の URIに関する応答の遅れや問題を見つけるのにも役立ちます。アクセスログを使用して、テスト環境のトラフィックパターンを解析、再生し、実際のユーザーの操作を模倣することもできます。アプリケーションやマーケットのトラブルシューティング、デバッグ、分析を行う際に、ログを活用する可能性は無限にあります。

14.2 エラーログの設定

問題

NGINXサーバーの問題をよりよく理解するために、エラーログを構成する必要があります。

解決法

`error_log`ディレクティブを使用してログパスとログレベルを定義します：

```
error_log /var/log/nginx/error.log warn;
```

error_logディレクティブにはパスが必要ですが、ログレベルはオプションで、デフォルトはerrorです。このディレクティブはifステートメントを除くすべてのコンテキストで有効です。利用可能なログレベルはdebug、info、notice、warn、error、crit、alert、emergです。ここに記載の順番は重大度の低いものから高いものとなっています。debugログレベルはNGINXの構成に--with-debugフラグがある場合にのみ利用可能です。

解説

エラーログは、構成ファイルが正しく機能していない場合に最初に確認するものです。ログは、FastCGIなどのアプリケーションサーバーによって生成されたエラーを見つける場合にも役立ちます。エラーログを使用して、ワーカー、メモリーの割り当て、クライアントIP、およびサーバーまでの接続をデバッグできます。エラーログはフォーマットできません。しかし、日付、レベル、メッセージという特定の形式をとります。

14.3 Syslogへの転送

問題

ログをSyslogリスナーに転送してログを一元化されたサービスに集約する必要があります。

解決法

error_logおよびaccess_logディレクティブを使用してログをSyslogリスナーに送信します：

```
error_log syslog:server=10.0.1.42 debug;

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

error_logおよびaccess_logディレクティブのsyslogパラメータにはコロンが続き、多くのオプションを追加できます。オプションには、接続するIP、DNS名、またはUnixソケットを示す必須のサーバーフラグのほか、facility、severity、tag、nohostnameなどのオプションのフラグが含まれます。serverオプションはポート番号とIPアドレスまたはDNS名をとります。しかし、デフォルトはUDP 514です。facilityオプションは、SyslogのRFC標準で定義されている23属性の1つであるログメッセージの機能属性を指し、デフォルト値はlocal7です。tagオプションは、メッセージに値のタグを付けます。デフォルトの値はnginxです。severityのデフォルトはinfoで、送信されたメッセージの重大度を示します。nohostnameフラグは無効にする機能で、ホスト名フィールドをSyslogメッセージヘッダーに追加し、値を取りません。nohostnameフラグは、Syslogメッセージヘッダーへのホスト名フィールドの追加を無効にする機能で、値を取りません。

解説

Syslogは、ログメッセージを送信し、それらのログを単一のサーバーまたは複数のサーバーに収集するための標準プロトコルです。一元化されたロケーションにログを送信すると、同一サービスの複数のインスタンスが複数のホストで実行されている場合のデバッグに役立ちます。これは集約ログと呼ばれます。ログを集約すると、1つの場所でログを表示できるため、サーバー間を移動して、タイムスタンプに沿って複数のログファイルをつなぎ合わせる必要がありません。一般的なログ集約スタックは、Elasticsearch、Logstash、Kibanaで、頭文字を取ってELKスタックとも呼ばれます。NGINXは`access_log`および`error_log`ディレクティブで、これらのログをSyslogリスナーに簡単にストリーミングできるようにします。

14.4 構成のデバッグ

問題

NGINXサーバーから予期しない結果が表示されています。

解決法

以下のヒントを参考に構成をデバッグします：

- NGINXは最も具体的に一致するルールを探してリクエストを処理します。そのため、構成を手作業でステップごとに実行することが少し難しくなりますが、これはNGINXにとって最も効率的な方法です。NGINXがリクエストを処理する方法について詳しくは、「関連項目」セクションのドキュメンテーションリンクを参照してください。

- デバッグログをオンにできます。デバッグログをオンにする場合、NGINXパッケージが`--with-debug`フラグを使用して構成および構築されていることを確認する必要があります。一般的なパッケージはそうなっていますが、独自に作成したパッケージや最小限のパッケージを実行している場合は、念のため`nginx -V`を実行して再確認することをお勧めします。デバッグが有効であることを確認したら、`error_log`ディレクティブのログレベルを`debug: error_log /var/log/nginx/error.log debug`に設定できます。

- 特定の接続のデバッグを有効にできます。`debug_connection`ディレクティブは、`events`コンテキスト内で有効で、パラメータとしてIPまたはCIDRの範囲を指定できます。このディレクティブは複数回宣言して、複数のIPアドレスやCIDRの範囲をデバッグ対象に追加できます。これは、すべての接続をデバッグすることで、パフォーマンスを落とすことなく運用中の問題をデバッグするときに役立ちます。

- 特定の仮想サーバーのみをデバッグできます。`error_log`ディレクティブは、メインの`http`、`mail`、`stream`、`server`および`location`のコンテキストで有効なので、`debug`ログレベルは、必要なコンテキストでのみ設定できます。

- コアダンプを有効にして、そこからバックトレースを取得できます。コアダンプはオペレーティングシステムまたはNGINX構成ファイルを介して有効にできます。詳しくは管

理者ガイドの「関連項目」セクションを参照してください。

- `rewrite_log`ディレクティブを on に設定（`rewrite_log on`）することで、rewrite 文で何が起きているかを記録できます。

解説

NGINXツールは幅広く活用でき、設定によって驚くべき成果を得ることができます。しかし、多くを実現する力には、自分の首を絞める事もあります。デバッグするときは、構成を通じて要求を追跡する方法を知っていることを確認してください。問題がある場合は、デバッグログレベルを追加してください。デバッグログは非常に冗長ですが、NGINXがリクエストに何をしているか、構成のどこが間違っているかを知る上で非常に役立ちます。

関連項目

NGINXリクエスト処理Request Processing

NGINXのデバッグ（管理者ガイド）

NGINX rewrite_logモジュールドキュメンテーション

14.5 リクエストトレース

問題

リクエストをエンドツーエンドで理解するには、NGINXログをアプリケーションログと関連付ける必要があります。

解決法

`request`特定変数を使用して、これをアプリケーションに渡して記録します：

```
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 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;
```

```
    }
  }
```

この例の構成では、traceという名前のlog_formatが設定され、ログでは変数$request_idが使用されています。$request_id変数はproxy_set_headerディレクティブを使用してアップストリームアプリケーションにも渡されます。これにより、アップストリームリクエストが作成された場合に、ヘッダーにリクエストIDが追加されます。リクエストIDは、レスポンスヘッダーにリクエストIDを設定するadd_headerディレクティブを介して、クライアントに返されます。

解説

NGINX Plus R10およびNGINXオープンソースバージョン1.11.0で利用可能になった、$request_id変数は、リクエストを一意に識別するために使用できる32個のランダムに生成された16進文字の文字列を提供します。この識別子をクライアントとアプリケーションに渡すことで、ログをリクエストと関連付けることができます。フロントエンドクライアントから、この一意の文字列をレスポンスヘッダーとして受け取り、それを使用して、対応するエントリをログで検索できます。ログとログの間に真のエンドツーエンドの関係を作成するには、このヘッダーを取得してアプリケーションログに記録するようにアプリケーションに指示する必要があります。この進歩により、NGINXはアプリケーションスタックを介してリクエストを追跡できるようにします。

関連項目

レシピ13.4「OpenTelemetry for NGINX」

第15章
パフォーマンス チューニング

15.0 はじめに

NGINXをチューニングするうちに芸術家になれます。あらゆるタイプのサーバーまたはアプリケーションのパフォーマンスチューニングは、環境、ユースケース、要件、関連する物理コンポーネントなど、さまざまな項目により常に異なりますが、関連する要因はこれらに限定されません。ボトルネック解消を主な目的としたチューニングを実践するのが一般的です。つまり、ボトルネックに達するまでテストし、ボトルネックを特定し、制限を調整して、目的のパフォーマンス要件に達するまで繰り返します。本章では、自動化されたツールを使用してテストし、結果を測定することにより、パフォーマンスチューニング時に測定を行うことをお勧めします。また、クライアントとアップストリームサーバーに対して接続を開いたままにし、オペレーティングシステムを調整することでより多くの接続を提供するための接続チューニングについても説明します。

15.1 ロードドライバーを使用したテストの自動化

問題

ロードドライバーを使用してテストを自動化し、テストに一貫性と再現性を実現する必要があります。

解決法

Apache JMeter、Locust、Gatling、または自社チームが標準化したものでも、HTTP負荷テストツールを使用します。Webアプリケーションで包括的なテストを実行する負荷テストツールの構成を作成します。サービスに対してテストを実行します。テストの実行結果から収集されたメトリクスを見直して、ベースラインを確立します。エミュレートされたユーザーの同時実行性をゆっくりと増やして、一般的な本番環境の使用法を模倣し、改善点を特定します。NGINXを調整し、目的の結果が得られるまでこのプロセスを繰り返します。

解説

自動テストツールを使用してテストを定義すると、NGINXを調整するときにメトリクスを構築するための一貫したテストが得られます。科学的に実施するために、テストを繰り返し、パフォーマンスの向上または低下を測定できなければなりません。NGINX構成を微調整してベースラインを確立する前にテストを実行することで、構成の変更によってパフォーマンスが向上したかどうかを測定するためのベースラインが得られます。加えた変更ごとに測定することで、パフォーマンスの向上が何から生じたのかを特定するのに役立ちます。

15.2 ブラウザでのキャッシュの制御

問題

クライアント側でキャッシュしてパフォーマンスを改善する必要があります。

解決法

クライアント側キャッシュ制御ヘッダーを使用します：

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

このlocationブロックはクライアントがCSSおよびJavaScriptファイルのコンテンツをキャッシュできるよう指定します。expiresディレクティブは、キャッシュされたリソースが1年後に無効になることをクライアントに指示します。add_headerディレクティブは、HTTP応答ヘッダー Cache-Controlをpublicという値とともに応答に追加します。この値は任意のキャッシュサーバーにリソースのキャッシュを許可します。privateを指定すると、クライアントのみが値のキャッシュを許可されます。

解説

キャッシュのパフォーマンスには多くの要因がありますが、ディスク速度がその上位に挙げられます。NGINX構成には、キャッシュのパフォーマンスを支援するために実行できることがたくさんあります。1つのオプションとして、クライアントが実際に応答をキャッシュし、NGINXに要求をまったく行わず、単に独自のキャッシュから応答するように、応答のヘッダーを設定できます。

15.3 クライアントに対して接続を開いたままにする

問題

単一の接続を介してクライアントが送信できる要求の数を増やし、アイドル状態の接続を維持する時間を延長する必要があります。

解決法

keepalive_requestsおよびkeepalive_timeoutディレクティブを使用して、単一の接続で実行できる要求の数を変更し、アイドル状態の接続を開いたままにしておくことができる時間を変更します。

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

keepalive_requestsディレクティブのデフォルトは100秒、keepalive_timeoutディレクティブのデフォルトは75秒です。

解説

最近のブラウザでは、FQDNごとに1つのサーバーへの複数の接続を開くことが許可されているため、通常、単一の接続を介したデフォルトの要求数でクライアントのニーズが満たされます。ドメインへの並列オープン接続の数は、通常10未満に制限されているため、この点に関しては、単一の接続を介した多くの要求が発生します。コンテンツ配信ネットワークで一般的に採用されているHTTP/1.1の秘訣は、コンテンツサーバーを指す複数のドメイン名を作成し、使用されるドメイン名をコード内で変更して、ブラウザーがより多くの接続を開くことができるようにすることです。これらの接続の最適化は、フロントエンドアプリケーションがバックエンドアプリケーションを継続的にポーリングして更新を確認する場合に役立つことがあります。より多くの要求を許可し、より長く開いたままにするオープン接続では、作成する必要のある接続の数が制限されるためです。

15.4 接続をアップストリームで開いたままにする

問題

パフォーマンスを向上させるために、再利用のためにアップストリームサーバーへの接続を開いたままにしておく必要があります。

解決法

upstreamコンテキストでkeepaliveディレクティブを使用して、再利用のためにアップストリームサーバーへの接続を開いたままにします。

```
    proxy_http_version 1.1;
    proxy_set_header Connection "";

    upstream backend {
      server 10.0.0.42;
      server 10.0.2.56;

      keepalive 32;
    }
```

upstreamコンテキストのkeepaliveディレクティブは、各NGINXワーカーに対して開いたままの接続のキャッシュをアクティブにします。このディレクティブは、ワーカーごとに開いたままにするアイドル接続の最大数を示します。upstreamブロックの上で使用されるプロキシモジュールディレクティブは、keepaliveディレクティブがアップストリームサーバー接続に対して正しく機能するために必要です。proxy_http_versionディレクティブは、プロキシモジュールにHTTPバージョン1.1を使用するように指示します。これにより、開いている間、単一の接続を介して複数の要求を連続して行うことができます。proxy_set_headerディレクティブは、プロキシモジュールにデフォルトのヘッダーのcloseを削除して、接続を開いたままにするように指示します。

解説

アップストリームサーバーへの接続を開いたままにして、接続の開始にかかる時間を節約し、ワーカープロセスがアイドル状態の接続を介して要求を行うために直接移動できるようにしたい場合には注意が必要です。開いている接続とアイドル状態の接続は異なるので、開いている接続の数は、keepaliveディレクティブで指定されている接続の数を超える可能性があります。keepalive接続の数は、アップストリームサーバーへの他の着信接続を許可できるよう少なく保つ必要があります。このNGINXチューニングトリックは、いくつかのサイクルを節約し、パフォーマンスを向上させることができます。

15.5 応答のバッファリング

問題

一時ファイルへの応答の書き込みを回避するために、アップストリームサーバーとクライアント間の応答をメモリにバッファする必要があります。

解決法

プロキシバッファ設定を調整して、NGINXメモリで応答本文をバッファできるようにします。

```
    server {
      proxy_buffering on;
      proxy_buffer_size 8k;
      proxy_buffers 8 32k;
      proxy_busy_buffer_size 64k;
```

```
    # ...
  }
```

proxy_buffering ディレクティブは on または off のいずれかで、デフォルトでは on です。proxy_buffer_size は、プロキシされたサーバーから応答および応答ヘッダーの最初の部分を読み取るために使用されるバッファのサイズを示し、プラットフォームによって異なりますが、デフォルトで 4k または 8k のいずれかです。proxy_buffers ディレクティブは、バッファの数とバッファのサイズの 2 つのパラメータを取ります。デフォルトでは、proxy_buffers ディレクティブは数字の 8、バッファサイズはプラットフォームによって 4k または 8k のいずれかです。proxy_busy_buffer_size ディレクティブは、ビジーになる可能性のあるバッファのサイズを制限し、応答が完全に読み込まれない間にクライアントに応答を送信します。ビジーバッファサイズは、デフォルトでプロキシバッファのサイズまたはバッファサイズの 2 倍です。プロキシバッファリングが無効の場合、NGINX はすでに要求本文の送信を開始しているため、障害が発生した場合に要求本文を次のアップストリームサーバーに送信することはできません。

解説

プロキシバッファは、応答本文の一般的なサイズに応じて、プロキシのパフォーマンスを大幅に向上させることができます。これらの設定をチューニングすると悪影響が生じる可能性があるため、返される平均の本文サイズを観察し、徹底的かつ繰り返しテストを実行する必要があります。非常に大きなバッファは、不要な場合に設定すると、NGINX ボックスのメモリを消費しつくす可能性があります。これらの設定は、大きな応答本文を返すことがわかっている特定の場所に設定して、最適なパフォーマンスを得ることができます。

関連項目

NGINX proxy_request_buffering モジュールドキュメンテーション

15.6 アクセスログのバッファリング

問題

システムに負荷がかかった時に、NGINX ワーカープロセスがブロックされるのを減らすために、ログをバッファする必要があります。

解決法

アクセスログのバッファサイズとフラッシュ時間を設定します：

```
  http {
    access_log /var/log/nginx/access.log main buffer=32k flush=1m gzip=1;
  }
```

access_log ディレクティブの buffer パラメータは、ディスクに書き込む前にログデータで満たすことができるメモリバッファのサイズを示します。access_log ディレクティブの flush パラメータは、ログがディスクに書き込まれる前にバッファに留まることができる最長時間を設定します。gzip を使用している場合、ログは記録される前に圧縮されます。レベル 1（最も速く、

低い圧縮）から9（最も遅く、最高の圧縮）が有効です。

解説

ログデータをメモリにバッファリングすることは、最適化に向けた小さな一歩かもしれません。しかし、要求の多いサイトやアプリケーションの場合、これにより、ディスクとCPUの使用量を大幅に調整できます。`buffer`パラメータを`access_log`ディレクティブに使用する場合、次のログエントリがバッファに収まらないと、ログがディスクに書き出されます。`flush`パラメータを`buffer`パラメータと組み合わせて使用すると、バッファ内のデータが指定時間よりも古い場合、ログがディスクに書き込まれます。バッファリングを有効にしてログをテーリングすると、`flush`パラメータで指定された時間まで遅延が発生する場合があります。

15.7 OSチューニング

問題

負荷の急騰やトラフィックの多いサイトに対処するために、より多くの接続を受け入れられるようにオペレーティングシステムをチューニングする必要があります。

解決法

*net.core.somaxconn*のカーネル設定を確認します。これは、NGINXが処理するためにカーネルが待ち行列に入れることができる接続の最大数です。これを512より大きく設定する場合は、NGINX構成の`listen`ディレクティブの`backlog`パラメータを一致するように設定する必要があります。このカーネル設定を調べる必要がある兆候は、カーネルログにそうするように明示されているかどうかです。NGINXは接続を非常に迅速に処理するため、ほとんどのユースケースでは、この設定を変更する必要はありません。

開いているファイル記述子の数を増やす必要が生じることの方が一般的です。Linuxでは、接続ごとにファイルハンドルが開かれます。したがって、NGINXをプロキシまたはロードバランサーとして使用している場合、アップストリームの接続が開いているため、NGINXが2つ開く可能性があります。多数の接続を提供するには、カーネルオプション`sys.fs.file_max`を使用してシステム全体でファイル記述子の制限を増やす必要があることがあります。または、システムユーザーの場合には、NGINXは*/etc/security/limits.conf*ファイルに従って稼働します。その際、`worker_connections`と`worker_rlimit_nofile`の数も増やす必要があります。これらの構成は両方とも、NGINX構成のディレクティブです。

より多くの接続をサポートするには、より多くのエフェメラルポートを有効にします。NGINXがリバースプロキシまたはロードバランサーとして稼働する場合、アップストリームのすべての接続は、リターントラフィック用の一時ポートを開きます。システム構成によっては、サーバーで開いているエフェメラルポートの最大数がない場合があります。これは、カーネル設定net. `ipv4.ip_local_port_range`を確認します。この設定は、ポートの下限と上限の範囲を指定するものです。通常、このカーネル設定は1024から65535で問題ありません。1024は登録済みのTCPポートが停止する場所であり、65535は動的ポートまたはエフェメラルポートが停止

する場所です。下限は、開いているリスニングサービスの最大ポートよりも高くする必要がある点に注意してください。

解説

オペレーティングシステムのチューニングは、多数の接続のチューニングを開始するときに最初に確認するところの1つです。特定のユースケースに合わせてカーネルに対して実行できる多くの最適化があります。しかし、カーネルの調整は気まぐれで行うべきではなく、変更が役立つことを確認するために、変更によるパフォーマンスの変化を測定する必要があります。前述のとおり、カーネルログのメッセージから、またはNGINXがエラーログにメッセージを明示的に記録することから、いつカーネルのチューニングを開始するべきかを把握できます。

索引

B

C

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

著者について

Derek DeJonghe はテクノロジーに情熱を抱いています。Web 開発、システム管理、ネットワーク構築分野における背景と経験こそ、彼の現代のWeb アーキテクチャ全般に関する理解と知識の源となっています。Derekは、サイトの信頼性とクラウドソリューションのエンジニアのチームを率い、ブティックコンサルティンググループで10年にわたり、自己修復型の自動スケーリングインフラストラクチャを作成しています。

現在は、新興企業によるクラウド環境と開発ツールのブートストラップを支援する専門インキュベーターの基盤作りに注力しています。Derekは、そのクラウドと開発リーダーシップの経験を活かし、市場開拓戦略を開発する起業家や、未来のテクノロジーへの道をコーディングするチームメンバーを支えています。回復力の高いクラウドアーキテクチャの実績を携え、Derekはお客様にとって一番有益なセキュリティと保守のためのクラウドデプロイメントのパイオニアを務めます。

奥付

NGINXクックブックの表紙の動物はユーラシアンオオヤマネコ (Lynx lynx) です。ヤマネコの中でも最大の種で、西ヨーロッパから中央アジアにかけた広い地域に生息しています。このヤマネコは、耳先からピンと立った黒い印象的な耳毛があり、顔はごわごわとした長い毛でおおわれています。その毛色は黄灰色から灰色がかった茶色で、下腹には白い毛が生えています。このオオヤマネコの体には暗い斑点が点在しており、北部に生息する亜種は南部の亜種よりも灰色で斑点が少ない傾向があります。

他のリンクスの仲間とは異なり、ユーラシアオオヤマネコは野生の鹿、ヘラジカ、さらには家畜の羊など、大型有蹄動物を捕食します。成猫は毎日2 ～ 5ポンドの肉を必要とし、最長1週間は1つの獲物を食べ続けます。

ユーラシアオオヤマネコは20世紀半ばに絶滅の危機に瀕しましたが、その後の保護活動により、このオオヤマネコの保護状況は「軽度懸念」まで回復しました。O'Reillyの書籍のカバーに描かれた動物の多くは絶滅の危機に瀕しています。その動物のすべてが世界にとって重要です。

カバーイラストは、ShawのZoologyの白黒の彫刻に基づいた、Karen Montgomery によるものです。本シリーズのデザインは、Edie Freedman、Ellie Volckhausen、Karen Montgomeryによるものです。カバーのフォントはGilroy SemiboldおよびGuardian Sansを使用しています。本文のフォントはAdobe Minion Proです。ヘッダーのフォントはAdobe Myriad Condensedです。コードで使用されているフォントはDalton MaagのUbuntu Monoです。