

NetScaler CPX 13.1

Contents

NetScaler CPXについて	6
サポートされる機能	6
サポートされているプラットフォーム	8
アーキテクチャとトラフィックフロー	9
NetScaler CPX で単一 IP アドレスが機能する方法	9
NetScaler CPXインスタンスから発信されるリクエストのトラフィックフロー	10
外部ネットワークから発信されるリクエストのトラフィックフロー	10
NetScaler CPXライセンス	12
NetScaler CPXライセンスの種類	12
NetScaler CPXインスタンスでサポートされるプールされたキャパシティ	13
NetScaler CPXライセンスはどのように機能しますか？	13
NetScaler CPX エクスプレス	14
NetScaler CPXライセンスモデル	14
NetScaler ConsoleサービスからNetScaler CPX向けに帯域幅ベースおよびvCPUベースのライセンスをプロビジョニングする	14
NetScaler ConsoleオンプレミスからNetScaler CPXの帯域幅ベースライセンスとvCPUベースライセンスをプロビジョニングする	16
デプロイメントをクリーンアップするためのコマンド	18
DockerでのNetScaler CPXインスタンスの展開	19
前提条件	19
Quay から NetScaler CPX イメージをダウンロードする	20
docker run コマンドを使用した NetScaler CPX インスタンスの展開	20
docker runコマンドを使用したNetScaler CPXの軽量バージョンの展開	25
Docker Compose を使用した NetScaler CPX インスタンスの展開	26
NetScaler CPX インスタンスを NetScaler Console に追加する	29
前提条件	29

環境変数を使用してNetScaler CPXインスタンスをNetScaler Consoleに追加する	30
Kubernetes ConfigMapを使用してNetScaler ConsoleにNetScaler CPXインスタンスを追加する	31
NetScaler CPX ライセンス アグリゲーター	35
NetScaler CPX License Aggregatorの主な利点	35
展開トポロジ	36
Helm チャートを使用して NetScaler CPX License Aggregator を展開する	36
前提条件	36
Helm チャートを使用した展開	37
Helm チャートのインストール	37
帯域幅プールライセンスを管理するための NetScaler CPX License Aggregator のインストール	37
vCPUライセンスを管理するためのNetScaler CPX License Aggregatorのインストール	38
複数のライセンスを管理するためのNetScaler CPX License Aggregatorのインストール	39
NetScaler CPX License Aggregatorからライセンスを取得するようにNetScaler CPXを構成する	40
NetScaler CPX の構成	43
コマンドラインインターフェイスを使用した NetScaler CPX インスタンスの構成	43
NetScaler CPX におけるデフォルト以外のパスワードの使用のサポート	43
NITRO API を使用した NetScaler CPX インスタンスの構成	44
ジョブを使用した NetScaler CPX インスタンスの構成	44
NetScaler CPXインスタンスでのAppFlowの構成	46
AppFlow®を有効にする	46
コレクターを指定する	47
AppFlowアクションの構成	48
AppFlowポリシーの構成	48
AppFlowポリシーのバインド	49
構成ファイルを使用してNetScaler CPXを構成する	50
静的構成ファイルの構造	50

NetScaler CPXにおける動的ルーティングのサポート	53
NetScaler CPXでREST APIを使用してBGPベースのルートヘルスインジェクションを構成する	53
Docker ロギングドライバーの構成	57
NetScaler CPXインスタンスのアップグレード	58
前提条件	58
NetScaler CPXインスタンスをアップグレードするには	59
NetScaler CPXインスタンスでのワイルドカード仮想サーバーの使用	61
プロキシとしてNetScaler CPXを展開して東西トラフィックフローを有効にする	63
シングルホストネットワークにNetScaler CPXを展開する	66
マルチホストネットワークにNetScaler CPXを展開する	68
トポロジ 1: NetScaler CPXとバックエンドサーバーが同じホスト上にある場合、クライアントが異なるネットワーク上にある場合	68
トポロジ 2: 物理サーバーとクライアントを備えた NetScaler CPX	70
トポロジ3：異なるホストにプロビジョニングされたNetScaler CPXとサーバー	71
ネットワークに直接アクセスしてNetScaler CPXを展開する	74
ConfigMapsを使用してKubernetesでNetScaler CPXを構成する	76
KubernetesノードのローカルDNSキャッシュとしてNetScaler CPXをデプロイする	82
ノード上にNetScaler CPXをDNSキャッシュとしてデプロイする	83
Kubernetes マスターノードでの設定	83
Kubernetes クラスター内のワーカーノードでの設定	88
Google Compute Engine に NetScaler CPX プロキシを展開する	90
前提条件	90
設定手順	90
ステップ1：GCE VMにDockerをインストールする	91
ステップ2：DockerインスタンスとのリモートAPI通信を構成する	102
ステップ3：NetScaler CPXイメージをインストールする	105
ステップ4：NetScaler CPXインスタンスを作成する	106
ステップ5：NetScaler Consoleを介してNetScaler CPXのライセンスを付与する	110

ステップ 6: NetScaler CPX で負荷分散サービスを設定し、構成を確認する	110
NGINX Web サーバーをインストールする	111
NetScaler CPX で負荷分散を設定し、両方のWebサービスへの負荷分散を確認する	113
NetScaler CPX のトラブルシューティング	128
fix	137

NetScaler CPXについて

NetScaler CPXは、Dockerホストにプロビジョニングできるコンテナベースのアプリケーションデリバリーコントローラーです。NetScaler CPXを使用すると、お客様はDockerエンジンの機能を活用し、コンテナベースのアプリケーションにNetScalerの負荷分散およびトラフィック管理機能を使用できます。1つまたは複数のNetScaler CPXインスタンスを、Dockerホスト上にスタンドアロンインスタンスとして展開できます。

NetScaler CPXインスタンスは、最大1 Gbpsのスループットを提供します。

NetScalerのコンテナ化されたフォームファクターとして、NetScaler CPXはKubernetes環境にうまく統合され、NetScalerクラウドネイティブソリューションの不可欠な部分を形成します。NetScalerクラウドネイティブソリューションは、Kubernetes環境でソフトウェアアプリケーションを迅速、俊敏、効率的に作成および提供するのに役立ちます。NetScalerクラウドネイティブソリューションを使用すると、Kubernetes環境のエンタープライズグレードの信頼性とセキュリティを確保できます。詳細については、「[NetScalerクラウドネイティブソリューション](#)」を参照してください。

このドキュメントは、Dockerとその仕組みに精通していることを前提としています。Dockerの詳細については、[<https://docs.docker.com>](https://docs.docker.com) のDockerドキュメントを参照してください。

サポートされる機能

NetScaler CPXは、次の機能をサポートしています。

- アプリケーションの可用性
 - L4負荷分散とL7コンテンツスイッチング
 - SSLオフロード
 - IPv6プロトコル変換
 - Microsoft SQL、MySQL負荷分散
 - AppExpertレート制御
 - サブスクリャー認識型トラフィックステアリング
 - サージ保護と優先キューイング
 - 動的ルーティングプロトコル

- アプリケーション高速化
 - クライアントおよびサーバーのTCP最適化
 - キャッシュリダイレクト
 - AppCompress
 - AppCache
- アプリケーションセキュリティ
 - L7リライトおよびレスポnder
 - L4 DoS防御
 - L7 DoS防御
 - Web Application Firewall (WAF)。NetScaler CPXは、他のNetScalerフォームファクターでサポートされているすべてのWAF機能をサポートしています。サポートされているWAF機能の詳細については、[NetScaler Web App Firewall](#)を参照してください。
 - アプリケーショントラフィックの認証、認可、および監査 (AAA)
- TCPプロトコル最適化
 - マルチパスTCP
 - バイナリ増加輻輳制御 (BIC) およびcubic TCP
- シンプルな管理性
 - Webロギング
 - AppFlow®
 - NetScaler® アプリケーションデリバリー管理
 - アクション分析
- アプリケーション最適化
 - 統合キャッシング
- BGPルーティングとルートヘルスインジェクション (RHI)

注:

NetScaler CPXアプライアンスに割り当てられたインターフェイス (Linuxホスト) では、Rx、Tx、GRO、GSO、LROなどのインターフェイス機能が無効になります。これらの機能は、NetScaler CPXアプライアンスが停止した後も無効のままです。また、そのようなインターフェイスのMTUは1500バイトに変更されます。

サポートされているプラットフォーム

NetScaler CPXは、以下のプラットフォームでサポートされています。

- クバネティス
- レッドハット オープンシフト
- パブリッククラウド
 - アマゾン エラスティック クバネティス サービス (EKS)
 - アジュール クバネティス サービス (AKS)
 - グーグル キューバネティス エンジン (GKE)
- ランチャー
- ピボタル コンテナ サービス (PKS)
- Docker バージョン 1.12 以降

アーキテクチャとトラフィックフロー

このセクションでは、NetScaler CPXブリッジモードのアーキテクチャとトラフィックフローについて説明します。NetScaler CPXはホストモードでも展開できます。

Dockerホスト上にNetScaler CPXインスタンスをプロビジョニングすると、DockerエンジンはCPXインスタンス上に仮想インターフェースeth0を作成します。このeth0インターフェースは、docker0ブリッジ上の仮想インターフェース（veth*）に直接接続されています。Dockerエンジンはまた、172.17.0.0/16ネットワーク内のNetScaler CPXインスタンスにIPアドレスを割り当てます。

CPXインスタンスのデフォルトゲートウェイはdocker0ブリッジのIPアドレスであり、これはNetScaler CPXインスタンスとのすべての通信がDockerネットワークを介して行われることを意味します。docker0ブリッジから受信したすべての着信トラフィックは、NetScaler CPXインスタンス上のeth0インターフェースによって受信され、NetScaler CPXパケットエンジンによって処理されます。

次の図は、Dockerホスト上のNetScaler CPXインスタンスのアーキテクチャを示しています。



NetScaler CPX で単一 IP アドレスが機能する方法

通常の NetScaler MPX または VPX アプライアンスが機能するには、少なくとも 3 つの IP アドレスが必要です。

- NetScaler IP (NSIP) アドレスと呼ばれる管理IPアドレス
- サーバーファームとの通信用サブネットIP (SNIP) アドレス
- クライアント要求を受け入れるための仮想サーバーIP (VIP) アドレス

NetScaler CPXインスタンスは、管理とデータトラフィックの両方に使用される1つの単一IPアドレスで動作します。

プロビジョニング中に、Dockerエンジンによって、NetScaler CPXインスタンスに1つのプライベートIPアドレス（単一IPアドレス）のみが割り当てられます。NetScalerインスタンスの3つのIP機能は、1つのIPアドレスに多重化されます。この単一IPアドレスは、NSIP、SNIP、およびVIPとして機能するために異なるポート番号を使用します。

次の図は、単一IPアドレスがNSIP、SNIP、およびVIPの機能を実行するためにどのように使用されるかを示しています。



NetScaler CPXインスタンスから発信されるリクエストのトラフィックフロー

Dockerは、NetScaler CPXインスタンスから発信されるトラフィックをdocker0 IPアドレスに誘導するために、IPテーブルとNATルールを暗黙的に構成します。

次の図は、NetScaler CPXインスタンスから発信されたpingリクエストが宛先に到達する仕組みを示しています。

ローカライズされた画像(/en-us/cpx/media/cpx-traffic-flow-from-cpx-instance.png)

この例では、pingリクエストはeth0インターフェース上のパケットエンジンによって、送信元IPアドレスをNetScaler CPXのIPアドレス（172.17.0.4）として送信されます。その後、Dockerホストはネットワークアドレス変換（NAT）を実行して、ホストのIPアドレス（192.68.x.x）を送信元IPアドレスとして追加し、リクエストを宛先（216.58.x.x）に送信します。宛先IPアドレスからの応答は、逆の同じパスをたどります。Dockerホストは応答に対してNATを実行し、eth0インターフェース上のNetScaler CPXインスタンスに応答を転送します。

外部ネットワークから発信されるリクエストのトラフィックフロー

外部通信を有効にするには、NetScaler CPXをプロビジョニングする際に、Dockerが80、22、およびその他の必要なポートを公開するようにパラメータを設定する必要があります。プロビジョニング中に公開するポートを設定していない場合は、これらのポートを利用可能にするためにDockerホスト上でNATルールを構成する必要があります。

インターネットから発信されたクライアントリクエストはDockerホストによって受信され、Dockerホストはポートアドレス変換（PAT）を実行して、パブリックIPアドレスとポートをNetScaler CPXインスタンスの単一のIPアドレスとポートにマッピングし、トラフィックをインスタンスに転送します。

次の図は、Dockerホストがポートアドレス変換を実行して、NetScaler CPXの単一のIPアドレスとポートにトラフィックを誘導する方法を示しています。

ローカライズされた画像(/en-us/cpx/media/cpx-traffic-flow-from-external-network.png)

この例では、DockerホストのIPアドレスは192.68.x.xで、NetScaler CPXインスタンスの単一IPアドレスは172.17.0.4です。NetScaler CPXインスタンスのSSHポート22は、Dockerホスト上のポート1100にマッピングされています。クライアントからのSSHリクエストは、IPアドレス192.68.x.xのポート1100で受信されます。Dockerホストはポートアドレス変換を実行して、このアドレスとポートをポート22上の単一IPアドレス172.17.0.4にマッピングし、クライアントリクエストを転送します。

NetScaler CPXライセンス

[NetScaler CPX](#)は、Dockerホストにプロビジョニングしてマイクロサービスベースのアプリケーションの負荷分散を行うことができる、コンテナベースのアプリケーションデリバリーコントローラーです。アプリケーションデリバリーのパフォーマンスを向上させるには、ライセンスされたCPXが必要です。NetScaler CPXはプールライセンスをサポートしています。NetScaler Consoleは、NetScaler CPXインスタンスのライセンスサーバーとして機能します。

NetScaler Consoleは、オンプレミスとクラウドサービスの両方で利用できます。NetScaler Consoleを使用して、すべてのNetScalerフォームファクターのプールされたキャパシティライセンスを管理できます。

オンプレミスのNetScaler Consoleの詳細については、[オンプレミスのNetScaler Console](#)を参照してください。NetScaler Consoleサービスの詳細については、[NetScaler Consoleサービス](#)を参照してください。

NetScaler CPXライセンスの種類

NetScaler CPXは、オンプレミスおよびクラウドベースの展開向けに、帯域幅および仮想CPU（コア）プールライセンスをサポートしています。

帯域幅プール: NetScaler CPXライセンスは、インスタンスによる帯域幅消費量に基づいて割り当てることができます。プールライセンスを使用すると、インスタンスに必要な帯域幅を確保し、必要以上に割り当てないことで、帯域幅の利用率を最大化できます。現在、NetScaler CPXはプレミアム帯域幅プールライセンスのみをサポートしています。

vCPUプール: 仮想CPU使用量ベースのライセンスでは、ライセンスは特定のNetScaler CPXインスタンスに付与されるCPUの数を指定します。そのため、NetScaler CPXは、ライセンスサーバーから仮想CPUの数にのみライセンスをチェックアウトできます。NetScaler CPXは、システムで実行されているCPUの数に応じてライセンスをチェックアウトします。vCPUプールの詳細については、[NetScaler仮想CPUライセンス](#)を参照してください。

NetScaler CPXインスタンスでサポートされるプールされたキャパシティ

製品	最大帯域幅	最小帯域幅	最小インスタンス数	最大インスタンス数	最小帯域幅単位
NetScaler CPX	40000 注: CPUの周波数、世代などによって異なります。	20 メガビット/秒	1	16	10 メガビット/秒

注: Citrixは現在、パブリッククラウドベースのサービス向けに、NetScaler CPXの消費ベースまたは従量課金ベースのライセンスモデルに取り組んでいます。準備が整い次第、パブリッククラウドマーケットプレイスで利用可能になります。

NetScaler CPXライセンスはどのように機能しますか？

NetScaler CPXプール容量: NetScaler CPXインスタンスが1つのインスタンスライセンスと必要な帯域幅のみをチェックアウトできる共通のライセンスプールです。インスタンスがこれらのリソースを必要としなくなると、それらを共通プールにチェックインし、これらのライセンスを必要とする他のインスタンスがリソースを利用できるようにします。

NetScaler CPXのチェックインおよびチェックアウトライセンス: NetScaler Consoleは、オンデマンドでNetScaler CPXインスタンスにライセンスを割り当てます。NetScaler CPXインスタンスは、プロビジョニング時にNetScaler Consoleからライセンスをチェックアウトし、インスタンスが破棄されたときにNetScaler Consoleにライセンスをチェックインできます。

NetScaler CPXの動作: 1 Gbpsまでのスループットをチェックアウトする単一のNetScaler CPXインスタンスは、インスタンスプールからのみチェックアウトし、帯域幅ライセンスプールからはチェックアウトしません。NetScaler CPXは、1 Gbpsの帯域幅使用量までこの方法で動作します。たとえば、CPXインスタンスが200 Mbpsの帯域幅を消費する場合、帯域幅プールではなく、ライセンスのインスタンスプールを使用します。ただし、NetScaler CPXインスタンスが1200 Mbpsのスループットを消費する場合、最初の1000 Mbpsはインスタンスプールから利用され、残りの200 Mbpsは帯域幅プールから消費されます。

NetScaler CPX エクスプレス

NetScaler CPX Expressは、オンプレミスおよびクラウド展開向けの無料のソフトウェアエディションです。[Quay](#)リポジトリからNetScaler CPXインスタンスをダウンロードすると、これはライセンスファイルを必要としないPOCで利用できるデフォルトの容量であり、以下の機能が付属しています。

- 20 Mbpsの帯域幅
- 最大250のSSLセッション
- 20 Mbps SSLスループット

パフォーマンス向上と実稼働環境への展開のためにアップグレードするには、NetScaler CPXインスタンスのライセンスを取得する必要があります。

NetScaler CPXライセンスモデル

NetScalerは、組織の要件を満たすために、NetScaler CPX向けにさまざまな製品ライセンスモデルを提供しています。vCPUまたは帯域幅、オンプレミスまたはクラウドなどのオプションを選択できます。

要件に基づいて、以下のいずれかのモデルを選択できます。

- ADMサービスからのNetScaler CPX向け帯域幅ベースのライセンス
- ADMサービスからのNetScaler CPX向けvCPUベースのライセンス
- オンプレミスADMからのNetScaler CPX向け帯域幅ベースのライセンス
- オンプレミスADMからのNetScaler CPX向けvCPUベースのライセンス

NetScaler ConsoleサービスからNetScaler CPX向けに帯域幅ベースおよびvCPUベースのライセンスをプロビジョニングする

NetScaler ConsoleサービスからNetScaler CPX向けに帯域幅ベースのライセンスとvCPUベースのライセンスをプロビジョニングするには、次の手順を実行します。

1. NetScaler Consoleをセットアップします。

NetScaler ConsoleサービスのセットアップがNetScalerエージェントで動作していることを確認します。NetScaler CPXライセンスが機能するには、NetScaler ConsoleサービスとNetScalerエージェントのアカウントが必要です。NetScaler ConsoleサービスとNetScalerエージェントのセットアップについては、「[NetScaler Console service](#)」を参照してください。

注: この手順では、ハイパーバイザー（オンプレミス）NetScalerエージェントのセットアップを使用します。次の画像では、`10.106.102.199` はNetScaler CPXのライセンスに使用されるオンプレミスエージェントです。



2. NetScalerインスタンスライセンスプールをNetScaler Consoleサービスに追加します。

ADMサービスで利用可能な帯域幅ライセンスのプールがあることを前提としています。

NetScaler Consoleへのライセンスファイルのアップロードについては、「[プールされた容量の構成](#)」を参照してください。次の画像では、`CNS_INST_200CC_Retail.lic` が帯域幅およびインスタンスライセンスプールとして使用されています。



3. KubernetesクラスターにNetScaler CPXインスタンスを展開します。NetScaler CPXインスタンスをライセンスするために、次の環境変数がNetScaler CPX YAMLファイルに追加されていることを確認してください。

NetScaler Consoleサービスからの帯域幅ベースのライセンスの場合、YAMLファイルで次の環境変数を指定します。

- name: "LS_IP" value: "10.105.158.166" //手順1に記載されているADMエージェントのIPアドレス
- name: "LS_PORT" value: "27000" // ADMライセンスサーバーがリッスンするポート
- name: "BANDWIDTH" value: "3000" //CPXに割り当てるMbps単位の容量
- name: "EDITION" value: "Standard" または "Enterprise" // Standard、Platinum、Enterpriseを含む特定のライセンスエディションを選択します。デフォルトではPlatinumが選択されます。

NetScaler ConsoleサービスからのvCPUベースのライセンスの場合、YAMLファイルで次の環境変数を指定します。

- name: "LS_IP" value: "10.102.216.173" // 手順1で指定されたADMエージェントのIPアドレス
- name: "LS_PORT" value: "27000" // ADMライセンスサーバーがリッスンするポート
- name: "CPX_CORES" value: "4" // 割り当てるコア数
- name: "PLATFORM" value: "CP1000" // コア数。チェックアウト数はコア数と同じです。

4. 次のコマンドを使用して `cpx-bandwidth-license-adm-service.yaml` ファイルをダウンロードします。

```
kubectl create namespace bandwidth

wget https://raw.githubusercontent.com/citrix/cloud-native-getting-started/master/cpx-licensing/manifest/cpx-bandwidth-license-adm-service.yaml
```

5. 編集したYAMLを以下のコマンドを使用してKubernetesクラスターにデプロイします。

```
kubectl create -f cpx-bandwidth-license-adm-service.yaml -n bandwidth
```

6. 以下のコマンドを使用して、インスタンス情報を確認するためにNetScaler CPXにログインします。

```
kubectl exec -it 'cpx-pod-ip-name' bash -n bandwidth
```

7. 指定されたNetScaler CPXインスタンスのライセンス情報を表示するには、以下のコマンドを実行します。

```
cli_script.sh "show licenseserver"

cli_script.sh "show capacity"
```

ADMサービスポータルで、割り当てられた帯域幅とvCPU容量を追跡できます。

NetScaler ConsoleオンプレミスからNetScaler CPXの帯域幅ベースライセンスとvCPUベースライセンスをプロビジョニングする

NetScaler ConsoleオンプレミスからNetScaler CPXに帯域幅ベースおよびvCPUベースのプロビジョニングを行うには、以下の手順を実行します。

1. NetScaler Consoleをセットアップします。

ADMオンプレミスセットアップが準備できていることを確認します。NetScaler CPXライセンス用のADMエージェント展開の有無にかかわらず、NetScaler Consoleオンプレミスが機能していることを確認してください。

NetScaler ConsoleオンプレミスとNetScalerエージェントのセットアップについては、[NetScaler Console service](#)を参照してください。

注: この例では、ADMオンプレミスに組み込まれたADMエージェントが使用されています。以下の画像では、エージェントが展開されていないことがわかります。



2. NetScalerインスタンスライセンスプールをADMオンプレミスに追加します。

ADMオンプレミスで利用可能な帯域幅ライセンスのプールがあることを前提としています。NetScaler Consoleへのライセンスファイルのアップロードについては、[Licensing](#)を参照してください。以下の画像では、`CNS_INST_200CC_Retail.lic` が帯域幅およびインスタンスライセンスプールとして使用されています。



以下の画像では、CP1000がvCPUライセンスプールとして使用されています。

vCPU ライセンスプール(/en-us/cpx/media/cpx-lic-vcpu-pool.png)

3. NetScaler CPX インスタンスを Kubernetes クラスターに展開します。NetScaler CPX インスタンスをライセンスするために、次の環境変数が NetScaler CPX YAML ファイルに追加されていることを確認してください。

NetScaler Console オンプレミスからの帯域幅ベースのライセンスの場合、YAML ファイルに次の環境変数を指定します。

- name: "LS_IP" value: "10.105.158.144" // ADM オンプレミスインスタンスの IP。ADM エージェントを展開している場合は、ステップ 1 で説明されているように、これがエージェントの IP アドレスになります。
- name: "LS_PORT" value: "27000" // ADM ライセンスサーバーがリッスンするポート
- name: "BANDWIDTH" value: "3000" // CPX に割り当てる Mbps 単位の容量

NetScaler Console オンプレミスからの vCPU ベースのライセンスの場合、YAML ファイルに次の環境変数を指定します。

- name: "LS_IP" value: "10.105.158.144" // ADM オンプレミスインスタンスの IP。ADM エージェントを展開している場合は、ステップ 1 で説明されているように、これがエージェントの IP アドレスになります。
- name: "LS_PORT" value: "27000" // ADM ライセンスサーバーがリッスンするポート
- name: "CPX_CORES" value: "4" // 割り当てるコア数
- name: "PLATFORM" value: "CP1000" // コア数。チェックアウト数はコア数と同じです。

4. 次のコマンドを使用して `cpx-bandwidth-license-adm-onprem.yaml` ファイルをダウンロードします。

```
kubectl create namespace bandwidth

wget https://raw.githubusercontent.com/citrix/cloud-native-getting-started/master/cpx-licensing/manifest/cpx-bandwidth-license-adm-onprem.yaml
```

5. 次のコマンドを使用して、編集した YAML を Kubernetes クラスターに展開します。

```
kubectl create -f cpx-bandwidth-license-adm-onprem.yaml -n bandwidth
```

6. 次のコマンドを使用して、インスタンス情報を確認するために NetScaler CPX にログインします。

```
kubectl exec -it <cpx-pod-ip-name> bash -n bandwidth
```

7. NetScaler CPX インスタンスのライセンス情報を表示するには、次のコマンドを実行します。

```
cli_script.sh "show licenseserver"

cli_script.sh "show capacity"
```

ADMオンプレミス ポータルで、割り当てられた帯域幅とvCPU容量を追跡できます。

デプロイメントをクリーンアップするためのコマンド

さまざまなYAMLデプロイメントをクリーンアップするには、次のコマンドを使用できます。

```
kubectl delete -f cpx-bandwidth-license-adm-service.yaml -n bandwidth

kubectl delete -f cpx-core-license-adm-service.yaml -n core

kubectl delete -f cpx-bandwidth-license-adm-onprem.yaml -n bandwidth

kubectl delete -f cpx-core-license-adm-onprem.yaml -n core

kubectl delete namespace bandwidth

kubectl delete namespace core
```

DockerでのNetScaler CPXインスタンスの展開

NetScaler CPXインスタンスは、QuayコンテナレジストリでDockerイメージファイルとして利用できます。インスタンスを展開するには、QuayコンテナレジストリからNetScaler CPXイメージをダウンロードし、`docker run` コマンドまたはDocker composeツールを使用してインスタンスを展開します。

前提条件

以下を確認してください。

- Dockerホストシステムには、少なくとも次のものがが必要です。
 - 1 CPU
 - 2 GB RAM

注

NetScaler CPXのパフォーマンスを向上させるには、NetScaler CPXインスタンスが起動する処理エンジンの数を定義できます。追加する処理エンジンごとに、Dockerホストに同等の数のvCPUとGB単位のメモリが含まれていることを確認してください。たとえば、4つの処理エンジンを追加する場合、Dockerホストには4つのvCPUと4 GBのメモリが含まれている必要があります。

- Dockerホストシステムは、Linux Ubuntuバージョン14.04以降を実行していること。
- ホストシステムにDockerバージョン1.12がインストールされていること。LinuxへのDockerのインストールについては、[Docker Documentation](#)を参照してください。
- Dockerホストがインターネットに接続されていること。

注: NetScaler CPXは、ubuntuバージョン16.04.5、カーネルバージョン4.4.0-131-genericで実行すると問題が発生します。そのため、NetScaler CPXをubuntuバージョン16.04.5、カーネルバージョン4.4.0-131-genericで実行することはお勧めしません。

注: 以下のkubeletおよびkube-proxyのバージョンにはいくつかのセキュリティ脆弱性があり、これらのバージョンでNetScaler CPXを使用することはお勧めしません。

- kubelet/kube-proxy v1.18.0-1.18.3
- kubelet/kube-proxy v1.17.0-1.17.6
- kubelet/kube-proxy <=1.16.10

この脆弱性を軽減する方法については、「[この脆弱性を軽減する](#)」を参照してください。

Quay から NetScaler CPX イメージをダウンロードする

Quay コンテナレジストリから NetScaler CPX イメージを `docker pull` コマンドを使用してダウンロードし、お使いの環境に展開できます。Quay コンテナレジストリから NetScaler CPX イメージをダウンロードするには、次のコマンドを使用します。

```
docker pull quay.io/citrix/citrix-k8s-cpx-ingress:13.0-xx.xx
```

たとえば、バージョン 13.0-64.35 をダウンロードする場合は、次のコマンドを使用します。

```
docker pull quay.io/citrix/citrix-k8s-cpx-ingress:13.0-64.35
```

NetScaler CPX イメージが Docker イメージにインストールされているかどうかを確認するには、次のコマンドを使用します。

```
root@ubuntu:~# docker images | grep 'citrix-k8s-cpx-ingress'
```

quay.io/citrix/citrix-k8s-cpx-ingress	13.0-64.35
952a04e73101	2 months ago 469 MB

[Quay コンテナレジストリ](#) から最新の NetScaler CPX イメージを展開できます。

docker run コマンドを使用した NetScaler CPX インスタンスの展開

ホスト上で、ホストにロードした NetScaler CPX Docker イメージを使用して、Docker コンテナに NetScaler CPX インスタンスをインストールできます。`docker run` コマンドを使用して、デフォルトの NetScaler CPX 構成で NetScaler CPX インスタンスをインストールします。

次の **docker run** コマンドを使用して、Docker コンテナに NetScaler CPX インスタンスをインストールします。

```
docker run -dt -P --privileged=true --net=host -e NS_NETMODE="HOST" -e
CPX_CORES=<number of cores> --name <container_name> --ulimit core=-1 -e
CPX_NW_DEV='<INTERFACES>' -e CPX_CONFIG='{ "YIELD": "NO" }' -e
LS_IP=<LS_IP_ADDRESS> -e LS_PORT=<LS_PORT> -e PLATFORM=CP1000 -v
<host_dir>:/cpx <REPOSITORY>:<TAG>
```

```
docker run -dt --privileged=true --net=host -e NS_NETMODE="HOST" -e
CPX_NW_DEV='eth1 eth2' -e CPX_CORES=5 -e CPX_CONFIG='{ "YIELD": "No" }' -e
LS_IP=10.102.38.134 -e PLATFORM=CP1000 -v /var/cpx:/cpx --name cpx_host
cpx:13.0-x.x
```

この例では、NetScaler CPX Docker イメージに基づいて **mycpx** という名前のコンテナを作成します。

-P パラメータは必須です。これは、NetScaler CPX Docker イメージによってコンテナで公開されているポートをマップするように Docker に指示します。つまり、ポート 9080、22、9443、および 161/UDP を、ユーザー定義の範囲からランダムに選択された Docker ホスト上のポートにマップします。このマッピングは、競合を回避するために行われます。同じ Docker ホスト上に複数の NetScaler CPX コンテナを後で作成する場合、ポートマッピングは動的であり、コンテナが起動または再起動されるたびに設定されます。ポートは次のように使用されます。

- 9080 は HTTP に使用されます
- 9443 は HTTPS に使用されます
- SSHに使用される22
- 161/UDPはSNMPに使用されます。

静的ポートマッピングが必要な場合は、**-p**パラメータを使用して手動で設定します。

--privileged=true オプションは、コンテナを特権モードで実行するために使用されます。NetScaler CPXをホストモードで展開している場合、NetScaler CPXにすべてのシステム特権を付与する必要があります。NetScaler CPXをシングルコアまたはマルチコアでブリッジモードで実行したい場合は、このオプションの代わりに **--cap-add=NET_ADMIN** オプションを使用できます。**--cap-add=NET_ADMIN** オプションを使用すると、NetScaler CPXコンテナを完全なネットワーク特権で実行できます。

**** --net=host** は、コンテナがホストネットワークスタックで実行され、すべてのネットワークデバイスにアクセスできることを指定する標準のdocker runコマンドオプションです。

注

NetScaler CPXをブリッジまたはnoneネットワークで実行している場合は、このオプションを無視してください。

-e NS_NETMODE="HOST" は、NetScaler CPXがホストモードで起動されることを指定できるNetScaler CPX固有の環境変数です。NetScaler CPXがホストモードで起動すると、NetScaler CPXへの管理アクセス用にホストマシン上に4つのデフォルトのiptablesルールが構成されます。以下のポートを使用します。

- HTTP用9995
- HTTPS用9996
- SSH用9997
- SNMP用9998

異なるポートを指定したい場合は、以下の環境変数を使用できます。

- -e NS_HTTP_PORT=
- -e NS_HTTPS_PORT=
- -e NS_SSH_PORT=
- -e NS_SNMP_PORT=

注

NetScaler CPXをブリッジネットワークまたはnoneネットワークで実行している場合は、この環境変数を無視してください。

-e CPX_CORES はオプションのNetScaler CPX固有の環境変数です。NetScaler CPXコンテナが起動する処理エンジンの数を定義することで、NetScaler CPXインスタンスのパフォーマンスを向上させることができます。

注

- 追加する処理エンジンごとに、Dockerホストに同等の数のvCPUとGB単位のメモリが含まれていることを確認してください。たとえば、4つの処理エンジンを追加したい場合、Dockerホストには4つのvCPUと4GBのメモリが含まれている必要があります。
- NetScaler CPXは1から16コアをサポートできます。

`-e EULA = yes` は必須のNetScaler CPX固有の環境変数であり、[The Citrix licensing system components](#) で入手可能なエンドユーザーライセンス契約 (EULA) を読み、理解したことを確認するために必要です。

`-e PLATFORM=CP1000` パラメータは、NetScaler CPXのライセンスタイプを指定します。

ホストネットワークでDockerを実行している場合、`-e CPX_NW_DEV` 環境変数を使用して、専用のネットワークインターフェースをNetScaler CPXコンテナに割り当てることができます。ネットワークインターフェースは空白で区切って定義する必要があります。定義したネットワークインターフェースは、NetScaler CPXコンテナをアンインストールするまでNetScaler CPXコンテナによって保持されます。NetScaler CPXコンテナがプロビジョニングされると、割り当てられたすべてのネットワークインターフェースがNetScalerネットワーク名前空間に追加されます。

注

NetScaler CPXをブリッジネットワークで実行している場合、コンテナネットワークを変更できます。たとえば、コンテナに別のネットワーク接続を設定したり、既存のネットワークを削除したりできます。その後、更新されたネットワークを使用するためにNetScaler CPXコンテナを再起動してください。

```
docker run -dt --privileged=true --net=host -e NS_NETMODE="HOST" -e
EULA=yes -e CPX_NW_DEV='eth1 eth2' -e CPX_CORES=5 -e PLATFORM=CP1000 --
name cpx_host cpx:13.0-x.x
```

`-e CPX_CONFIG` は、NetScaler CPXコンテナのスループットパフォーマンスを制御できるNetScaler CPX固有の環境変数です。NetScaler CPXが処理する受信トラフィックを受け取らない場合、アイドル時にCPUを解放するため、スループットパフォーマンスが低下します。このようなシナリオでは、`CPX_CONFIG` 環境変数を使用してNetScaler CPXコンテナのスループットパ

パフォーマンスを制御できます。`CPX_CONFIG` 環境変数には、JSON形式で以下の値を指定する必要があります。

- アイドルシナリオでNetScaler CPXコンテナにCPUを解放させたい場合は、`{"YIELD" : "Yes"}` を定義します。
- NetScaler CPXコンテナがアイドル状態のシナリオでCPUの解放を回避し、高いスループット性能を得たい場合は、`{"YIELD" : "No"}` を定義します。

```
docker run -dt --privileged=true --net=host -e NS_NETMODE="HOST" -e
EULA=yes -e CPX_CORES=5 -e CPX_CONFIG='{"YIELD":"No"}' -e PLATFORM=CP1000
--name cpx_host cpx:13.0-x.x
```

```
docker run -dt --privileged=true --net=host -e NS_NETMODE="HOST" -e
EULA=yes -e CPX_CORES=5 -e CPX_CONFIG='{"YIELD":"Yes"}' -e PLATFORM=CP1000
--name cpx_host cpx:13.0-x.x
```

`-v` パラメータは、NetScaler CPXマウントディレクトリ `/cpx` のマウントポイントを指定するオプションパラメータです。マウントポイントとは、ホスト上のディレクトリで、そこに `/cpx` ディレクトリをマウントします。`/cpx` ディレクトリには、ログ、構成ファイル、SSL証明書、コアダンプファイルが保存されます。この例では、マウントポイントは `/var/cpx` で、NetScaler CPXマウントディレクトリは `/cpx` です。

ライセンスを購入した場合、または評価ライセンスをお持ちの場合は、ライセンスをライセンスサーバーにアップロードし、`-e LS_IP=<LS_IP_ADDRESS> -e LS_PORT=<LS_PORT>` パラメータを使用してdocker runコマンドでライセンスサーバーの場所を指定できます。この場合、EULAに同意する必要はありません。

```
docker run -dt --privileged=true --net=host -e NS_NETMODE="HOST" -e
CPX_CORES=5 -e CPX_CONFIG='{"YIELD":"No"}' -e LS_IP=10.102.38.134 -e
PLATFORM=CP1000 --name cpx_host cpx:13.0-x.x
```

ここで、：

- `LS_IP_ADDRESS` はライセンスサーバーのIPアドレスです。
- `LS_PORT` はライセンスサーバーのポートです。

システムで実行中のイメージと標準ポートにマッピングされているポートは、次のコマンドを使用して表示できます: `docker ps`

docker runコマンドを使用したNetScaler CPXの軽量バージョンの展開

NetScalerは、ランタイムメモリの消費量が少ないNetScaler CPXの軽量バージョンを提供しています。NetScaler CPXの軽量バージョンは、サービスメッシュ展開でサイドカーとして展開できます。

NetScaler CPXの軽量バージョンは、以下の機能をサポートしています。

- アプリケーションの可用性
 - L4ロードバランシングとL7コンテンツスイッチング
 - SSLオフロード
 - IPv6プロトコル変換
- アプリケーションセキュリティ
 - L7 リライトとレスポnder
- シンプルな管理性
 - Webロギング
 - AppFlow®

NetScaler CPXの軽量バージョンをインスタンス化するには、`Docker run` コマンドの実行中に `NS_CPX_LITE` 環境変数を設定します。

```
docker run -dt -P --privileged=true -e NS_CPX_LITE=1 -e EULA=yes --name  
<container_name> --ulimit core=-1 <REPOSITORY>:<TAG>
```

次の例は、NetScaler CPXイメージに基づいて軽量コンテナを作成する方法を示しています。

```
docker run -dt -P --privileged=true -e NS_CPX_LITE=1 -e EULA=yes --name  
lightweight --ulimit core=-1 cpx:latest
```

デフォルトでは、NetScaler CPXの軽量バージョンでは `newnslog` を使用したロギングは無効になっています。これを有効にするには、NetScaler CPXの軽量バージョンを起動する際に、`NS_ENABLE_NEWNSLOG` 環境変数を1に設定する必要があります。

次の例は、NetScaler CPXの軽量バージョンをデプロイする際に `newnslog` を使用したロギングを有効にする方法を示しています。

```
docker run -dt --privileged=true --ulimit core=-1 -e EULA=yes -e  
NS_CPX_LITE=1 -e NS_ENABLE_NEWNSLOG=1 cpx:<tag>
```

注:

CPXの軽量バージョンはシングルコアのみをサポートします (CPX_CORES=1)。

Docker Compose を使用した NetScaler CPX インスタンスの展開

Docker の Compose ツールを使用して、単一の NetScaler CPX インスタンスまたは複数の NetScaler CPX インスタンスをプロビジョニングできます。Docker Compose を使用して NetScaler CPX インスタンスをプロビジョニングするには、まずコンポーズファイルを作成する必要があります。このファイルには、NetScaler CPX イメージ、NetScaler CPX インスタンス用に開きたいポート、および NetScaler CPX インスタンスの特権が指定されます。

重要

ホストに Docker Compose ツールがインストールされていることを確認してください。

複数の NetScaler CPX インスタンスをプロビジョニングするには:

1. コンポーズファイルを作成します。その中で:

- `<service-name>` は、プロビジョニングするサービスの名前です。
- `image:<repository>:<tag>` は、NetScaler CPXイメージのリポジトリとバージョンを示します。
- `privileged: true` は、NetScaler CPXインスタンスにすべてのルート権限を提供します。
- `cap_add` は、NetScaler CPXインスタンスにネットワーク権限を提供します。
- `<host_directory_path>` は、NetScaler CPXインスタンスのためにマウントするDockerホスト上のディレクトリを示します。
- `<number_processing_engine>` は、NetScaler CPXインスタンスが起動する処理エンジンの数です。追加の処理エンジンごとに、Dockerホストに同等の数のvCPUとGB単位のメモリが含まれていることを確認してください。たとえば、4つの処理エンジンを追加する場合、Dockerホストには4つのvCPUと4GBのメモリが含まれている必要があります。

コンポーズファイルは通常、次のような形式に従います。

```
<container_name>:

  image: <repository>:<tag>

  ports:

    - 9443

    - 22

    - 9080

    - 161/udp

  tty: true

  cap_add:

    - NET_ADMIN

  ulimits:

    core: -1

  volumes:

    - <host_directory_path>:/cpx

  environment:

    - EULA=yes

    - CPX_CORES=<number_processing_engine>

    - CPX_CONFIG='{"YIELD":"Yes"}'
```

```
CPX_0:

  image: quay.io/citrix/citrix-k8s-cpx-ingress:13.1-37.38

  ports:
```

```
- 9443

- 22

- 9080

- 161/udp

tty: true

cap_add:

- NET_ADMIN

ulimits:

core: -1

volumes:

- /root/test:/cpx

environment:

- CPX_CORES=2

- EULA=yes
```

NetScaler CPX インスタンスを NetScaler Console に追加する

これらのインスタンスを管理および監視したい場合は、Docker ホストにインストールされている NetScaler CPX インスタンスを NetScaler Application Delivery Management (ADM) ソフトウェアに追加する必要があります。

インスタンスは、ADMを初めてセットアップするとき、または後で追加できます。

インスタンスを追加するには、インスタンスプロファイルを作成し、各インスタンスのホスト名またはIPアドレス、あるいはIPアドレスの範囲を指定する必要があります。このインスタンスプロファイルには、NetScaler Consoleに追加するインスタンスのユーザー名とパスワードが含まれています。インスタンスタイプごとに、デフォルトプロファイルが利用可能です。たとえば、`ns-root-profile` はNetScalerインスタンスのデフォルトプロファイルです。このプロファイルは、デフォルトのADC管理者資格情報によって定義されます。インスタンスのデフォルトの管理者資格情報を変更した場合は、それらのインスタンスのカスタムインスタンスプロファイルを定義できます。インスタンスが検出された後にインスタンスの資格情報を変更した場合は、インスタンスプロファイルを編集または作成してから、インスタンスを再検出する必要があります。

前提条件

以下を確認してください。

- Citrix XenServerにNetScaler Consoleソフトウェアをインストール済みであること。詳細については、「[NetScaler Console Documentation](#)」を参照してください。
- DockerホストにNetScaler CPXインスタンスをインストール済みであること。

NetScaler CPXインスタンスをADMに追加するには：

1. Webブラウザで、**NetScaler Application Delivery Management** のIPアドレスを入力します（例：`http://192.168.100.1`）。
2. ユーザー名とパスワードのフィールドに、管理者資格情報を入力します。デフォルトの管理者資格情報は**nsroot**と**nsroot**です。
3. **Networks** > **Instances** > **NetScaler** に移動し、**CPX**タブをクリックします。
4. **追加**をクリックして、NetScaler Consoleに新しいCPXインスタンスを追加します。

5. NetScaler CPXの追加ページが開きます。次のパラメータの値を入力します。

- a. CPXインスタンスを追加するには、CPXインスタンスの到達可能なIPアドレス、またはCPXインスタンスがホストされているDockerコンテナのIPアドレスのいずれかを指定します。
- b. CPXインスタンスのプロファイルを選択します。
- c. インスタンスをデプロイするサイトを選択します。
- d. エージェントを選択します。
- e. オプションとして、インスタンスにキーと値のペアを入力できます。キーと値のペアを追加すると、後でインスタンスを簡単に検索できるようになります。

CPXインスタンスを追加(/en-us/cpx/media/add-cpx-instances.png)

6. OKをクリックします。

注

インスタンスを再検出する場合は、ネットワーク > インスタンス > NetScaler > CPX を選択し、再検出するインスタンスを選択してから、アクションの選択ドロップダウンリストから再検出をクリックします。

環境変数を使用してNetScaler CPXインスタンスをNetScaler Consoleに追加する

環境変数を使用してNetScaler CPXインスタンスをNetScaler Consoleに追加することもできます。インスタンスを追加するには、NetScaler CPXインスタンスに対して次の環境変数を構成する必要があります。

- **NS_MGMT_SERVER** - ADM IPアドレス/FQDN
- **HOST** - ノードIPアドレス
- **NS_HTTP_PORT** - ノード上のマッピングされたHTTPポート
- **NS_HTTPS_PORT** - ノード上のマッピングされたHTTPSポート
- **NS_SSH_PORT** - ノード上のマッピングされたSSHポート
- **NS_SNMP_PORT** - ノード上のマッピングされたSNMPポート
- **NS_ROUTABLE** - (NetScaler CPXポッドのIPアドレスは外部からルーティングできません。)
- **NS_MGMT_USER** - ADMユーザー名

- `NS_MGMT_PASS` – ADMパスワード

以下は、NetScaler CPXインスタンスをNetScaler Consoleに追加するための `docker run` コマンドの例です。

```
docker run -dt --privileged=true -p 9080:9080 -p 9443:9443 -p 9022:22 -p 9161:161 -e EULA=yes -e NS_MGMT_SERVER=abc-mgmt-server.com -e HOST=10.1.1.1 -e NS_HTTP_PORT=9080 -e NS_HTTPS_PORT=9443 -e NS_SSH_PORT=9022 -e NS_SNMP_PORT=9161 -e NS_ROUTABLE=0 --ulimit core=-1 -name test cpx:latest
```

Kubernetes ConfigMapを使用してNetScaler ConsoleにNetScaler CPXインスタンスを追加する

NetScaler CPXは、Kubernetes ConfigMapを介してボリュームマウントされたファイルを使用することで、NetScaler Consoleへの登録をサポートします。この登録方法を有効にするには、NetScaler CPXは、ConfigMapとSecretを介してボリュームマウントされたいくつかのファイルとともに指定されるいくつかの環境変数を必要とします。

以下に、必要な環境変数とその説明を示します。

- `NS_HTTP_PORT` - ノード上のマッピングされたHTTPポートを指定します。
- `NS_HTTPS_PORT` - ノード上のマッピングされたHTTPSポートを指定します。
- `NS_SSH_PORT` - ノード上のマッピングされたSSHポートを指定します。
- `NS_SNMP_PORT` - ノード上のマッピングされたSNMPポートを指定します。

リストされている環境変数とは別に、NetScaler CPXは登録する必要があるADMエージェントに関する情報を必要とします。この情報には、ADMエージェントのIPアドレスまたはFQDNの詳細と資格情報が含まれます。NetScaler CPXは、ボリュームマウントされたファイルからこの情報を取得します。IPアドレスまたはFQDNを含むConfigMapは、NetScaler CPXインスタンスのファイルシステムにファイルとしてマウントされます。ADMエージェントの資格情報を含むKubernetesシークレットも、NetScaler CPXインスタンスのファイルシステムにファイルとしてマウントされます。登録に必要なすべての情報が揃うと、NetScaler CPXはADMエージェントへの登録を試みます。

以下は、ConfigMapとSecretがファイルとしてマウントされたNetScaler CPX YAMLファイルスニペットの例です。

```
...

env:

- name: "EULA"

  value: "yes"

- name: "NS_HTTP_PORT"

  value: "9080"

- name: "NS_HTTPS_PORT"

  value: "9443"

- name: "NS_SSH_PORT"

  value: "22"

- name: "NS_SNMP_PORT"

  value: "161"

- name: "KUBERNETES_TASK_ID"

  value: ""

...

volumeMounts:

  ...

  - mountPath: /var/admininfo/server/

    name: adm-agent-config

  - mountPath: /var/admininfo/credentials/

    name: adm-agent-user

  ...
```

```

volumes:

...

- name: adm-agent-config

  configMap:

    name: adm-agent-config

- name: adm-agent-user

  secret:

    secretName: adm-secret

```

上記の例では、`adm-agent-config` という名前のConfigMapと `adm-agent-user` というシークレットが使用されています。以下は、必要なConfigMapとSecretを作成する例です。

ConfigMap: ConfigMapは、`adm_reg_envs` という名前のファイルから作成されます。このファイルには、ADMエージェントのIPアドレスまたはFQDNを次の形式で記述する必要があります。

```
NS_MGMT_SERVER=adm-agent
```

上記の形式では、`adm-agent` はNetScaler CPXインスタンスを登録する必要があるADMエージェントのFQDNです。

次のコマンドを使用してConfigMapを作成します。

```
kubectl create configmap adm-agent-config --from-file=adm_reg_envs
```

注: ファイル名には `adm_reg_envs` 変数が含まれている必要があり、`/var/adminfo/server/` のパスにマウントする必要があります。

Secret: 次のコマンドを使用してKubernetesシークレットを作成します。次のコマンドでは、`user123` がADMエージェントのユーザー名、`pass123` がパスワードです。

```
kubectl create secret generic adm-secret --from-
literal=NS_MGMT_USER=user123 --from-literal=NS_MGMT_PASS=pass123
```

NetScaler CPXインスタンスは、クラスターにADMエージェントをデプロイする前であっても、必要な環境変数とボリュームマウントされたファイルを使用してKubernetesクラスターにデプロイできます。ADMエージェントをデプロイする前にNetScaler CPXインスタンスをデプロイした場合、NetScaler CPXはADMエージェントがデプロイされるまで登録を試行し続けます。ADMエージェントがデプロイされると、NetScaler CPXインスタンスは、環境変数とボリュームマウントされたファイルを通じて提供された構成データを使用してADMエージェントに登録します。これにより、構成情報を含むNetScaler CPXの再デプロイを回避できます。

すでにADMエージェントに登録されているNetScaler CPXインスタンスは、構成の変更後に別のADMエージェントへの登録を動的に変更できます。このためには、すでにデプロイされているNetScaler CPXのConfigMapとSecretの構成情報を更新できます。ConfigMapが作成されたファイルを新しいADMエージェントのIPアドレスまたはFQDNで更新し、古いConfigMapを削除してから、新しいConfigMapを作成する必要があります。同様に、既存のシークレットを削除し、新しいADMエージェントの資格情報で新しいシークレットを作成する必要があります。

NetScaler CPX ライセンス アグリゲーター

現在、NetScaler CPXはNetScaler Consoleサーバーからライセンスを取得しています。

Kubernetes環境では、NetScaler CPXは動的に起動または停止できます。NetScaler CPXが予期せず停止した場合、NetScaler Consoleサーバーがライセンスを再利用するのに数分かかります。

NetScaler Consoleサーバーは、NetScaler CPXが停止するとすぐにそのようなライセンスを再利用できる必要があります。そうすることで、同じライセンスを別の起動中のNetScaler CPXに割り当てることができます。また、何らかの理由でNetScaler Consoleサーバーに到達できない場合、クラスター内の新しいNetScaler CPXにライセンスを付与することはできません。

NetScaler CPX License Aggregatorは、NetScalerが提供するKubernetesサービスです。このサービスは、Kubernetesクラスター内でNetScaler CPXライセンスのローカルプロバイダーとして機能します。Kubernetesクラスターに展開されたNetScaler CPX License Aggregatorサービスは、NetScaler CPXとADMライセンスサーバー間の仲介役として機能し、NetScaler CPXと割り当てられたライセンスを追跡できます。NetScaler CPX License Aggregatorサービスを使用すると、NetScaler ConsoleサーバーはNetScaler CPXが停止するとすぐにライセンスを再利用できます。

Kubernetesクラスターでは、NetScaler CPX License Aggregatorサービスは、サイドカーおよびスタンドアロン展開の両方でNetScaler CPXをサポートします。

注記:

NetScaler CPX License Aggregatorを使用したライセンス付与には、NetScaler CPX 13.1-30.x以降が必要です。NetScaler CPX License Aggregatorは、以前のバージョンのNetScaler CPXのライセンスをサポートしていません。

NetScaler CPX License Aggregatorの主な利点

NetScaler CPX License Aggregatorを使用する主な利点は次のとおりです。

- スケーラビリティ: NetScaler Consoleライセンスサーバーは、最大10000のNetScaler CPX展開のみをサポートできます。NetScaler CPX License Aggregatorサービスの導入により、各KubernetesクラスターはNetScaler Consoleライセンスサーバーの単一クライアントとして機

ことができます。したがって、単一のNetScaler Consoleライセンスサーバーで多数のNetScaler CPXをスケーリングできます。

- リソースの最適化: NetScaler CPX License Aggregatorサービスは、クラスター全体のライセンス機能もサポートしており、必要に応じてNetScaler Consoleサーバーからライセンスをチェックアウトすることもできます。NetScaler CPX License Aggregatorは、ライセンスをNetScaler Consoleサーバーに返却できます。NetScaler CPX License Aggregatorは、NetScaler CPXの不適切な終了を処理し、設定された待機期間後にそのようなNetScaler CPXからライセンスを再利用できます。

展開トポロジ

次の図は、Kubernetesクラスター内でのNetScaler CPX License Aggregatorの展開を示しています。



この図では、次のようになります。

- **CPX** はNetScaler CPXを意味します
- **CPX License Aggregator** は NetScaler CPX License Aggregator を意味します

このサンプル展開では、NetScaler CPX License Aggregator サービスは、NetScaler CPX および NetScaler エージェントとともに Kubernetes クラスター内に展開されます。NetScaler CPX License Aggregator サービスは、NetScaler CPX と NetScaler エージェント間の仲介役として機能し、クラスター内のすべての NetScaler CPX を監視し、それらのライセンスを管理します。

Helm チャートを使用して NetScaler CPX License Aggregator を展開する

前提条件

以下の前提条件が適用されます。

- Kubernetes バージョン 1.16 以降が必要です。
- Helm バージョン 3.x 以降が必要です。
- NetScaler CPX のライセンスを持つライセンスサーバーの IP アドレスを取得する必要があります。
- NetScaler CPX License Aggregator の Redis データベースにパスワードを指定する必要があります。データベースパスワードは Kubernetes シークレットを使用して指定でき、シークレットを作成するには次のコマンドを使用できます。

```
kubectl create secret generic dbsecret --from-  
literal=password=&lt;custom-password>
```

Helm チャートを使用した展開

NetScaler CPX ライセンスの種類に応じて、Helm チャートを使用して NetScaler CPX License Aggregator を展開するには、次の手順を実行します。さまざまな種類の NetScaler CPX ライセンスの詳細については、「[NetScaler CPX ライセンス](#)」を参照してください。

Helm チャートのインストール

次のコマンドを使用して、NetScaler CPX License Aggregator Helm チャートリポジトリを追加します。

```
helm repo add Citrix https://citrix.github.io/citrix-helm-charts/
```

帯域幅プールライセンスを管理するための NetScaler CPX License Aggregator のインストール

お持ちの NetScaler CPX プールライセンスの種類に応じて、次のいずれかのコマンドを使用します。これらのコマンドでは、`my-release` がリリース名として使用されます。

注記:

デフォルトでは、Helmチャートは推奨されるRBACロールとロールバインディングをインストールします。

プラチナ帯域幅ライセンスの場合：

```
helm install my-release citrix/cpx-license-aggregator --set  
licenseServer.address=&lt;License-Server-IP-or-  
FQDN>,redis.secretName=&lt;Kubernetes-Secret-for-DB-  
password>,licenseAggregator.username=&lt;unique-ID-for-  
CLA>,licenseInfo.instanceQuantum=&lt;QUANTUM>,licenseInfo.instanceLowWater  
mark=&lt;LOW WATERMARK>,licenseInfo.bandwidthPlatinumQuantum=&lt;QUANTUM-  
in-Mbps>,licenseInfo.bandwidthPlatinumLowWatermark=&lt;LOW WATERMARK-in-  
Mbps>
```

エンタープライズ帯域幅エディションの場合：

```
helm install my-release citrix/cpx-license-aggregator --set
licenseServer.address=<License-Server-IP-or-
FQDN>,redis.secretName=<Kubernetes-Secret-for-DB-
password>,licenseAggregator.username=<unique-ID-for-
CLA>,licenseInfo.instanceQuantum=<QUANTUM>,licenseInfo.instanceLowWater
mark=<LOW
WATERMARK>,licenseInfo.bandwidthEnterpriseQuantum=<QUANTUM-in-
Mbps>,licenseInfo.bandwidthEnterpriseLowWatermark=<LOW WATERMARK-in-
Mbps>
```

スタンダード帯域幅エディションの場合：

```
helm install my-release citrix/cpx-license-aggregator --set
licenseServer.address=<License-Server-IP-or-
FQDN>,redis.secretName=<Kubernetes-Secret-for-DB-
password>,licenseAggregator.username=<unique-ID-for-
CLA>,licenseInfo.instanceQuantum=<QUANTUM>,licenseInfo.instanceLowWater
mark=<LOW WATERMARK>,licenseInfo.bandwidthStandardQuantum=<QUANTUM-
in-Mbps>,licenseInfo.bandwidthStandardLowWatermark=<LOW WATERMARK-in-
Mbps>
```

これらのコマンドは、デフォルト設定でNetScaler CPX License AggregatorをKubernetesクラスターにデプロイします。インストール時にパラメーターを設定できます。詳細については、インストール時に設定できる必須およびオプションのパラメーターを一覧表示している[Helm chart](#) [GitHub リポジトリ](#)の**NetScaler CPX License Aggregator configuration**セクションを参照してください。

vCPUライセンスを管理するためのNetScaler CPX License Aggregatorのインストール

お持ちのNetScaler CPX vCPUライセンスの種類に応じて、以下のいずれかのコマンドを使用してください。これらのコマンドでは、`my-release` がリリース名として使用されます。

注記: デフォルトでは、Helmチャートは推奨されるRBACロールとロールバインディングをインストールします。

プラチナvCPUエディションの場合：

```
helm install my-release citrix/cpx-license-aggregator --set
licenseServer.address=&lt;License-Server-IP-or-
FQDN>,redis.secretName=&lt;Kubernetes-Secret-for-DB-
password>,licenseAggregator.username=&lt;unique-ID-for-
CLA>,licenseInfo.vcpuPlatinumQuantum=&lt;QUANTUM>,licenseInfo.vcpuPlatinum
LowWatermark=&lt;LOW WATERMARK>
```

エンタープライズvCPUエディションの場合：

```
helm install my-release citrix/cpx-license-aggregator --set
licenseServer.address=&lt;License-Server-IP-or-
FQDN>,redis.secretName=&lt;Kubernetes-Secret-for-DB-
password>,licenseAggregator.username=&lt;unique-ID-for-
CLA>,licenseInfo.vcpuEnterpriseQuantum=&lt;QUANTUM>,licenseInfo.vcpuEnterp
riseLowWatermark=&lt;LOW WATERMARK>
```

スタンダードvCPUエディションの場合：

```
helm install my-release citrix/cpx-license-aggregator --set
licenseServer.address=&lt;License-Server-IP-or-
FQDN>,redis.secretName=&lt;Kubernetes-Secret-for-DB-
password>,licenseAggregator.username=&lt;unique-ID-for-
CLA>,licenseInfo.vcpuStandardQuantum=&lt;QUANTUM>,licenseInfo.vcpuStandard
LowWatermark=&lt;LOW WATERMARK>
```

複数のライセンスを管理するためのNetScaler CPX License Aggregatorのインストール

NetScaler CPX License Aggregatorで複数の種類のライセンスを管理する必要がある場合、それらのライセンスの関連する引数をHelmコマンドで指定する必要があります。

例：

`pooled platinum bandwidth edition` および `vCPU platinum edition` ライセンス用の NetScaler CPX License Aggregatorを展開するには:

```
helm install demo citrix/cpx-license-aggregator --set
licenseServer.address=&lt;License-Server-IP-or-
FQDN>,redis.secretName=&lt;Kubernetes-Secret-for-DB-
password>,licenseAggregator.username=&lt;unique-ID-for-
```

```
CLA>,licenseInfo.instanceQuantum=&lt;QUANTUM>,licenseInfo.instanceLowWater  
mark=&lt;LOW WATERMARK>,licenseInfo.bandwidthPlatinumQuantum=&lt;QUANTUM-  
in-Mbps>,licenseInfo.bandwidthPlatinumLowWatermark=&lt;LOW WATERMARK-in-  
Mbps>,licenseInfo.vcpuPlatinumQuantum=&lt;QUANTUM>,licenseInfo.vcpuPlatinu  
mLowWatermark=LOW WATERMARK>
```

NetScaler CPX License Aggregatorからライセンスを取得するように NetScaler CPXを構成する

NetScaler CPXのライセンス供与にNetScaler CPX License Aggregatorを使用する場合、環境変数 `CLA` をNetScaler CPX展開YAMLに指定する必要があります。

NetScaler CPX License Aggregatorにアクセスするために使用する `ipaddress` または `domainname` を、この環境変数に次のように指定する必要があります:

```
env:  
  
- name: "CLA"  
  
  value: "192.0.2.2"
```

または

```
env:  
  
- name: "CLA"  
  
  value: "local-cla.org"
```

NetScaler CPX YAMLには、以下の環境変数も指定する必要があります。

- `POD_NAME` : Podの名前を指定します。Podの名前は環境変数としてNetScaler CPXに公開されます。
- `POD_NAMESPACE` : Podの名前空間を指定します。Podの名前空間は環境変数としてNetScaler CPXに公開されます。
- `Bandwidth` : NetScaler CPXに割り当てる帯域幅をMbpsで指定します。
- `Edition` : ライセンスエディションを指定します。サポートされている値には、Standard、Platinum、Enterpriseが含まれます。
- `CPX_CORES` : NetScaler CPXで実行するコア数を指定します。

さまざまなNetScaler CPXライセンスオプションの詳細については、「[NetScaler CPXライセンス](#)」を参照してください。

以下に、これらの環境変数を使用した構成例を示します:

```
- name: POD_NAME

valueFrom:

  fieldRef:

    apiVersion: v1

    fieldPath: metadata.name

- name: POD_NAMESPACE

valueFrom:

  fieldRef:

    apiVersion: v1

    fieldPath: metadata.namespace

- name: "BANDWIDTH"

value: 1000

- name: "CPX_CORES"

value: 1

- name: "EDITION"

value: PLATINUM
```

NetScaler CPX YAMLに以下のラベルも追加する必要があります:

```
labels:

  adc: citrix®
```

NetScaler CPX License Aggregatorの展開例については、「[NetScaler CPX License Aggregator: サンプル展開](#)」を参照してください。

NetScaler CPX の構成

NetScaler CPX インスタンスは、Linux Docker ホスト経由で CLI プロンプトにアクセスするか、NetScaler NITRO API を使用して構成できます。

コマンドラインインターフェイスを使用した NetScaler CPX インスタンスの構成

次の図に示すように、Docker ホストにアクセスし、インスタンスの SSH プロンプトにログオンします。NetScaler CPX インスタンスにログオンするためのデフォルトの管理者資格情報は root/linux です。

ローカライズされた画像(/en-us/cpx/media/cpx-configure-using-linux-1.png)

インスタンスのコマンドラインプロンプトを使用して CLI コマンドを実行するには、次のコマンドを入力します: `cli_script.sh "\<command\>"`

例:

ローカライズされた画像(/en-us/cpx/media/cpx-configure-using-linux-2.png)

インスタンスプロンプトからログアウトするには、**logout** と入力します。

NetScaler CPX におけるデフォルト以外のパスワードの使用のサポート

NetScaler CPX は、root アカウントにデフォルト以外のパスワードを使用することをサポートしています。これは `nsroot` です。NetScaler CPX がデプロイされると、デフォルトのパスワードが生成され、ユーザーに割り当てられます。このデフォルトパスワードは、SSH ユーザーの `root` および `nsroot` にも更新されます。このデフォルトパスワードは手動で変更できます。`root` および `nsroot` ユーザーアカウントのデフォルトの SSH パスワードも手動でリセットできます。Citrix は、システムのセキュリティを維持するために、このパスワードを手動で変更することを推奨しています。

パスワードをリセットすると、新しいパスワードは NITRO API 通信および `cli_script.sh` の実行に使用されます。デフォルトの root アカウントパスワードは、NetScaler CPX ファイルシステムの `/var/deviceinfo/random_id` ファイルにプレーンテキストで保存されます。

資格情報を使用して `cli_script.sh` を実行するには、次の構文を使用します。

```
cli_script.sh "<command>" ":<user>:<password>"
```

例えば、ユーザー `nsroot` とパスワード `Citrix123` を使用して IP アドレスを表示するために `cli_script.sh` を実行するには、次を使用します。

```
cli_script.sh "show ns ip" ":nsroot:Citrix123"
```

NITRO API を使用した NetScaler CPX インスタンスの構成

NetScaler NITRO API を使用して、NetScaler CPX インスタンスを構成できます。

Nitro API を使用して NetScaler CPX インスタンスを構成するには、Web ブラウザで次のように入力します:

```
http://<host_IP_address>:<port>/nitro/v1/config/<resource-type>
```

Nitro API を使用して統計情報を取得するには、Web ブラウザで次のように入力します:

```
http://<host\_IP\_address>:\<port>/nitro/v1/stat/\<resource-type>
```

NITRO API の使用方法の詳細については、「[REST Web サービス](#)」を参照してください。

NetScaler CPX の場合は、`netScaler-ip-address` が記載されている箇所で `CPX IP address:port` を使用します。

ジョブを使用した NetScaler CPX インスタンスの構成

NetScaler Console でジョブを作成および実行することで、NetScaler CPX インスタンスを構成できます。構成テンプレートの構成を使用したり、他のデバイスで利用可能な構成を抽出したり、テキストファイルに保存されている構成を使用したりできます。また、別のインスタンスの構成ユーティリティを使用して行われた構成を記録することもできます。NetScaler Console は、NetScaler CPX インスタンスで使用する対応する CLI コマンドを表示します。構成を選択したら、構成をロードする **NetScaler CPX インスタンス** を選択し、変数値を指定してジョブを実行する必要があります。

ジョブを使用して NetScaler CPX インスタンスを構成するには:

1. 管理者資格情報を使用して NetScaler Console にログインします。
2. **Networks** > **Configuration Jobs** に移動し、**Create Job** をクリックします。
3. 必要な値を指定し、構成ソースを選択します。実行したいコマンドを入力することもできます。



4. 構成を実行する NetScaler CPX インスタンスを選択し、**Next** をクリックします。



5. 実行設定を指定し、「完了」をクリックしてNetScaler CPXインスタンスでコマンドを実行します。設定を保存して後で実行する場合は、「**保存して終了**」をクリックします。



NetScaler CPXインスタンスでのAppFlowの構成

NetScaler CPXインスタンスでAppFlow機能を構成して、ウェブページのパフォーマンスデータ、フローおよびユーザーセッションレベルの情報、アプリケーションパフォーマンス監視と分析に必要なデータベース情報を収集できます。これらのデータレコードはNetScaler Consoleに送信され、そこですべてのアプリケーションのリアルタイムおよび履歴レポートを表示できます。

AppFlowを構成するには、まずAppFlow機能を有効にする必要があります。次に、フローレコードの送信先となるコレクターを指定します。その後、構成されたコレクターのセットであるアクションを定義します。次に、1つ以上のポリシーを構成し、各ポリシーにアクションを関連付けます。このポリシーは、関連付けられたアクションにフローレコードが送信されるリクエストを選択するようにNetScaler CPXに指示します。最後に、各ポリシーをグローバルに、または特定の仮想サーバーにバインドして、有効にします。

テンプレートの更新間隔を指定し、`httpURL`、`httpCookie`、および `httpReferer` 情報の書き出しを有効にするために、AppFlowパラメーターをさらに設定できます。各コレクターで、NetScaler CPXのIPアドレスをエクスポートのアドレスとして指定する必要があります。

構成ユーティリティは、ユーザーがポリシーとアクションを定義するのに役立つツールを提供します。NetScaler CPXが特定のフローのレコードをコレクターのセット（アクション）にどのようにエクスポートするかを正確に決定します。コマンドラインインターフェイスは、コマンドラインを好む経験豊富なユーザー向けに、対応するCLIベースのコマンドセットを提供します。

レコードを監視する前に、NetScaler CPXインスタンスをNetScaler Consoleに追加する必要があります。NetScaler ConsoleへのNetScaler CPXインスタンスの追加の詳細については、

「[NetScaler Consoleを使用してNetScaler CPXインスタンスをインストールする](#)」を参照してください。

AppFlow®を有効にする

AppFlow機能を使用するには、まず有効にする必要があります。

コマンドラインインターフェイスを使用してAppFlow機能を有効にするには：

次のコマンドを実行します。

```
enable ns feature AppFlow
```

```
enable ns mode ulfd
```

コレクターを指定する

コレクターは、NetScalerによって生成されたAppFlowレコードを受信します。AppFlowレコードを送信するには、少なくとも1つのコレクターを指定する必要があります。デフォルトでは、コレクターはUDPポート4739でIPFIXメッセージをリッスンします。コレクターを構成するときに、デフォルトポートを変更できます。

コマンドラインインターフェイスを使用してコレクターを指定するには：

コレクターを追加するには、次のコマンドを使用します。

```
add appflow collector <name> -IPAddress <ipaddress> -port  
<port_number> -netprofile <netprofile_name> -Transport Logstream
```

構成を確認するには、次のコマンドを使用します。

```
show appflow collector <name>
```

コマンドラインインターフェイスを使用して複数のコレクターを指定するには：

次のコマンドを使用して、同じデータを複数のコレクターに追加して送信します。

```
add appflow collector <collector1> -IPAddress <IP> -Transport  
Logstream  
  
add appflow collector <collector2> -IPAddress <IP> -Transport  
Logstream  
  
add appflow action <action> -collectors <collector1> <collector2>  
-Transport Logstream
```

```
add appflow policy <policy> true <action> -Transport Logstream

bind lbvserver <lbvserver> -policy <policy> -priority <priority>
-Transport Logstream
```

AppFlowアクションの構成

AppFlowアクションは、関連するAppFlowポリシーが一致した場合にフローレコードが送信される設定済みコレクターです。

AppFlowアクションを構成するには、次のコマンドを使用します。

```
add appflow action <name> --collectors <string> ... [-
clientSideMeasurements (Enabled|Disabled) ] [-comment <string>]
```

構成を確認するには、次のコマンドを使用します。

```
show appflow action
```

AppFlowポリシーの構成

AppFlowアクションを構成したら、次にAppFlowポリシーを構成する必要があります。AppFlowポリシーは、1つ以上の式で構成されるルールに基づいています。

コマンドラインインターフェイスを使用してAppFlowポリシーを構成するには：

コマンドプロンプトで、次のコマンドを入力してAppFlowポリシーを追加し、構成を確認します。

```
add appflow policy <name> <rule> <action>

show appflow policy <name>
```

AppFlowポリシーのバインド

ポリシーを有効にするには、NetScaler CPXを通過するすべてのトラフィックに適用されるように、グローバルにバインドする必要があります。

コマンドラインインターフェイスを使用してAppFlowポリシーをグローバルにバインドするには：

AppFlowポリシーをグローバルにバインドするには、次のコマンドを使用します。

```
bind appflow global <policyName> <priority>
[<gotoPriorityExpression [-type <type>] [-invoke (<labelType>
<labelName>)]
```

次のコマンドを使用して構成を確認します。

```
show appflow global
```

構成ファイルを使用してNetScaler CPXを構成する

コマンドラインインターフェイス (`cli_script.sh`)、NITRO API、またはNetScaler Console構成ジョブを使用してNetScaler CPXを構成する代わりに、NetScaler CPXインスタンスの展開中に静的構成ファイルを使用してNetScaler CPXを構成できます。

NetScaler CPXコンテナの展開時に、静的構成ファイルを入力ファイルとして提供できます。NetScaler CPXコンテナの起動中、コンテナは静的構成ファイルで指定された構成に基づいて構成されます。この構成には、NetScaler固有の構成と、NetScaler CPXコンテナで動的に実行できるbashシェルコマンドが含まれます。

静的構成ファイルの構造

前述のとおり、NetScaler CPXが展開されると、静的構成ファイルで指定された構成に基づいて構成されます。

静的構成ファイルは、`.conf` ファイルであり、`#NetScaler Commands` と `#Shell Commands` の2つのタグが含まれています。`#NetScaler Commands` タグの下には、NetScaler CPXでNetScaler固有の構成を設定するためのすべてのNetScalerコマンドを追加する必要があります。`#Shell Commands` タグの下には、NetScaler CPXで実行したいシェルコマンドを追加する必要があります。

NetScaler CPXコンテナの展開中、NetScalerコマンドとシェルコマンドは、構成ファイルで指定された順序でコンテナ上で実行されます。

重要:

- タグは構成ファイル内で複数回繰り返すことができます。
- タグは大文字と小文字を区別しません。
- 構成ファイルは、コンテナのファイルシステム内の `/etc` ディレクトリに `cpx.conf` ファイルとして存在する必要があります。
- 構成ファイルにはコメントを含めることもできます。コメントの前に「#」文字を追加する必要があります。
- 構成ファイルを使用してNetScaler CPXコンテナを展開する際に障害が発生した場合、その障害はコンテナ内の `ns.log` ファイルに記録されます。
- NetScaler CPXコンテナを再起動すると、構成ファイルがコンテナに再適用されます。

#NetScaler Commands

```
add lb vserver v1 http 1.1.1.1 80
```

```
add service s1 2.2.2.2 http 80
```

```
bind lb vserver v1 s1
```

#Shell Commands

```
touch /etc/a.txt
```

```
echo "this is a" > /etc/a.txt
```

```
#NetScaler Commands
```

```
add lb vserver v2 http
```

```
#Shell Commands
```

```
echo "this is a 1" >> /etc/a.txt
```

```
#NetScaler Commands
```

```
add lb vserver v3 http
```

```
#This is a test configuration file
```

NetScaler CPXコンテナをインストールし、構成ファイルに基づいてNetScaler CPXコンテナを動的に構成するには、`docker run` コマンドの `-v` オプションを使用して静的構成ファイルをマウントします。

```
docker run -dt --privileged=true -e EULA=yes --ulimit core=-1 -v /tmp/  
cpx.conf:/etc/cpx.conf --name mycpx store/citrix/citrixadccpx:13.0-x.x
```

NetScaler CPXにおける動的ルーティングのサポート

NetScaler CPXはBGP動的ルーティングプロトコルをサポートしています。動的ルーティングプロトコルの主な目的は、仮想サーバーにバインドされたサービスの健全性に基づいて、仮想サーバーのIPアドレスをアドバタイズすることです。これにより、アップストリームルーターは、地理的に分散された仮想サーバーへの複数のルートの中から最適なルートを選択できます。

NetScaler CPXの非デフォルトパスワードについては、[NetScaler CPXの構成](#)ドキュメントの `Support for using a non-default password in NetScaler CPX` セクションを参照してください。

単一ホストネットワークでは、クライアント、サーバー、およびNetScaler CPXインスタンスは、同じDockerホスト上にコンテナとして展開されます。すべてのコンテナはdocker0ブリッジを介して接続されます。この環境では、NetScaler CPXインスタンスは、同じDockerホスト上にコンテナとしてプロビジョニングされたアプリケーションのプロキシとして機能します。NetScaler CPXホストネットワークモードの展開については、[ホストネットワークモード](#)を参照してください。

次の図は、単一ホストトポロジを示しています。



このトポロジでは、仮想サーバーが構成され、BGPを使用して（サービスの健全性に基づいて）アップストリームネットワークまたはルーターにアドバタイズされます。

ブリッジネットワークモードの単一DockerホストでNetScaler CPXにBGPを構成するには、次の手順を実行します。

NetScaler CPXでREST APIを使用してBGPベースのルートヘルスインジェクションを構成する

1. 次のコマンドを使用して、NetScaler CPXイメージからコンテナを作成します。

```
docker run -dt --privileged=true -p 22 -p 80 -p 161 -e EULA=yes --ulimit core=-1 cpx: <tag>
```

例：

```
docker run -dt --privileged=true -p 22 -p 80 -p 161 -e EULA=yes --  
ulimit core=-1 cpx:12.1-50.16
```

2. 次のコマンドを使用してコンテナにログインします。

```
docker exec -it <container id> bash
```

3. 次のコマンドを使用してBGP機能を有効にします。

```
cli_script.sh "enable ns feature bgp"
```

4. `show ns ip` コマンドを使用してNSIPを取得します。

```
cli_script.sh "show ns ip"
```

5. 次のコマンドを使用して仮想サーバーを追加します。

```
cli_script.sh "add lb vserver <vserver_name> http <VIP>  
<PORT>"
```

6. サービスを追加し、仮想サーバーにサービスをバインドします。

7. 次のコマンドを使用して、VIP の `hostroute` を有効にします。

```
cli_script.sh "set ns ip <VIP> -hostroute enabled"
```

コンテナからログアウトし、ホストからポート 9080 の NSIP に BGP NITRO コマンドを送信します。

8. BGP ルーターを構成します。

たとえば、次のように構成する場合：

```
router bgp 100  
  
Neighbour 172.17.0.2 remote-as 101  
  
Redistribute kernel
```

コマンドを次のように指定します。

```
curl -u username:password http://&lt;NSIP>:9080/nitro/v1/config/ -X
POST --data 'object={"routerDynamicRouting": {"bgpRouter" :
{"localAS":100, "neighbor": [{ "address": "172.17.0.2", "remoteAS":
101}], "afParams":{"addressFamily": "ipv4", "redistribute":
{"protocol": "kernel"}}}}}'
```

9. 次の NITRO コマンドを使用して、学習した BGP ルートを PE にインストールします。

```
curl -u username:password http://&lt;NSIP>:9080/nitro/v1/config/ --data
'object={"params":{"action":"apply"},"routerDynamicRouting":
{"commandstring" : "ns route-install bgp"}}'
```

10. 次の NITRO コマンドを使用して、BGP 隣接状態を確認します。

```
curl -u username:password http://&lt;NSIP>:9080/nitro/v1/config/
routerDynamicRouting/bgpRouter
```

出力例：

```
root@ubuntu:~# curl -u username:password http://172.17.0.3:9080/nitro/
v1/config/routerDynamicRouting/bgpRouter

{ "errorcode": 0, "message": "Done", "severity": "NONE",
"routerDynamicRouting":{"bgpRouter":[{"localAS": 100, "routerId":
"172.17.0.3", "afParams": [ { "addressFamily": "ipv4" },
{ "addressFamily": "ipv6" } ], "neighbor": [ { "address": "172.17.0.2",
"remoteAS": 101, "ASOriginationInterval": 15, "advertisementInterval":
30, "holdTimer": 90, "keepaliveTimer": 30, "state": "Connect",
"singlehopBfd": false, "multihopBfd": false, "afParams":
[ { "addressFamily": "ipv4", "activate": true }, { "addressFamily":
"ipv6", "activate": false } ]
```

11. 次のコマンドを使用して、BGP を介して学習されたルートがパケットエンジンにインストールされていることを確認します。

```
cli_script.sh "show route"
```

12. 次のコマンドを使用して構成を保存します。

```
cli_script.sh "save config"
```

動的ルーティング構成は、`/nsconfig/ZebOS.conf` ファイルに保存されます。

Docker ロギングドライバーの構成

Docker には、実行中のコンテナから情報を取得するのに役立つ「ロギングドライバー」と呼ばれるロギングメカニズムが含まれています。NetScaler CPX コンテナが生成するログを Docker ロギングドライバーに転送するように設定できます。Docker ロギングドライバーの詳細については、[ロギングドライバーの構成](#)を参照してください。

デフォルトでは、NetScaler CPX コンテナによって生成されたすべてのログは、Docker ホスト上の `/cpx/log/ns.log` ファイルに保存されます。docker run コマンドを使用して NetScaler CPX コンテナを起動するときに、`--log-driver` オプションを使用して、生成されたすべてのログを Docker ロギングドライバーに転送するように設定できます。ロギングドライバーに設定可能なパラメータがある場合は、`--log-opt <NAME>=<VALUE>` オプションを使用してそれらを設定できます。

以下の例では、NetScaler CPX コンテナは、syslog をロギングドライバーとして使用して、生成されたすべてのログを転送するように構成されています。

```
docker run -dt --privileged=true --log-driver syslog --log-opt syslog-  
address=udp://10.106.102.190:514 -e EULA=yes --ulimit core=-1 --name test  
store/citrix/cpx:12.1-48.13
```

同様に、以下の例では、NetScaler CPX コンテナは、Splunk をロギングドライバーとして使用して、生成されたすべてのログを転送するように構成されています。

```
docker run -dt --privileged=true --log-driver=splunk --log-opt splunk-  
token=176FCEBF-4CF5-4EDF-91BC-703796522D20 --log-opt splunk-url=https://  
splunkhost:8088 -e EULA=yes --ulimit core=-1 --name test store/citrix/  
cpx:12.1-48.13
```

NetScaler CPXインスタンスのアップグレード

NetScaler CPXインスタンスは、シャットダウンし、同じマウントポイントに最新バージョンをインストールしてから、古いインスタンスを削除することでアップグレードできます。マウントポイントとは、ホスト上の **/cpx** ディレクトリをマウントするディレクトリのことです。

例えば、既存のNetScaler CPXインスタンスの **/cpx** ディレクトリをホストの **/var/cpx** ディレクトリにマウントする場合、マウントポイントは **/var/cpx** であり、NetScaler CPXのマウントディレクトリは **/cpx** です（以下に示すとおり）。

```
root@ubuntu:~# docker run -dt -e EULA=yes --name mycpx -v /var/cpx:/cpx --ulimit core=-1 cpx:13.0-x.x
```

前提条件

以下があることを確認してください。

- 既存のNetScaler CPXインスタンスの **/cpx** ディレクトリをマウントしたホストディレクトリの詳細。ホストディレクトリに関する情報を表示するには、**<containerName>** がNetScaler CPXコンテナの名前である **docker inspect <containerName>** コマンドを使用できます。

コマンドの出力には、ボリュームを含むコンテナ構成の詳細が示されます。「**Mounts**」エントリの「**Source**」サブエントリには、ホスト上のホストディレクトリの場所が表示されます。



- 最新のNetScaler CPX Dockerイメージファイルをダウンロードし、NetScaler CPX Dockerイメージをロードします。イメージをロードするには、Dockerイメージファイルを保存したディレクトリに移動します。イメージをロードするには、**docker load -i <image_name>** コマンドを使用します。NetScaler CPXイメージがロードされたら、**docker images** コマンドを入力してイメージに関する情報を表示できます。

```
root@ubuntu:~# docker load -i cpx-13.0-x.x.gz
```

```
root@ubuntu:~# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
cpx	13.0-x.x	2e97aadf918b	43 hours ago	414.5 MB

NetScaler CPXインスタンスをアップグレードするには

1. 既存のNetScaler CPXインスタンスを停止するには、`docker stop <containerName>` コマンドを入力します。ここで、`<containerName>` はNetScaler CPXインスタンスの名前です。

```
root@ubuntu:~# docker stop mycpx

mycpx
```

2. `docker run` コマンドを使用して、ホストにロードしたNetScaler CPXイメージから最新のNetScaler CPXインスタンスを展開します。既存のNetScaler CPXインスタンスに使用したのと同じマウントポイント（例: `/var/cpx:/cpx`）にインスタンスを展開するようにしてください。

```
root@ubuntu:~# docker run -dt -P -e CPX_CORES=1 --name latestcpx --
ulimit core=-1 -e EULA=yes -v /var/cpx:/cpx --cap-add=NET_ADMIN
cpx:13.0-x.x
```

展開されたNetScaler CPXインスタンスが最新バージョンであることを確認するために、`docker ps` コマンドを入力できます。

```
...
```

```
root@ubuntu:~# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS		PORTS	NAMES

```
ead12ec4e965    cpx:13.0-x.x    "/bin/sh -c 'bash -C "    5 seconds ago
Up 5 seconds    22/tcp, 80/tcp, 443/tcp, 161/udp    latestcpx
```

```
...
```

- 正しいNetScaler CPXインスタンスを展開したことを確認したら、古いインスタンスを削除するために、**docker rm** `<containerName>` コマンドを入力します。

```
root@ubuntu:~# docker rm mycpx
```

```
mycpx
```

NetScaler CPXインスタンスでのワイルドカード仮想サーバーの使用

NetScalerインスタンスをプロビジョニングすると、Dockerエンジンによって、NetScaler CPXインスタンスに1つのプライベートIPアドレス（単一IPアドレス）のみが割り当てられます。

NetScalerインスタンスの3つのIP機能は、1つのIPアドレスに多重化されます。この単一IPアドレスは、NSIP、SNIP、VIPとして機能するために異なるポート番号を使用します。

Dockerエンジンによって割り当てられる単一IPアドレスは動的です。ロードバランシング (LB) またはコンテンツスイッチング (CS) 仮想サーバーを、単一IPアドレスまたは127.0.0.1 IPアドレスを使用して追加します。127.0.0.1を使用して作成された仮想サーバーは、ワイルドカード仮想サーバーと呼ばれます。デフォルトでは、ワイルドカード仮想サーバーを作成すると、NetScaler CPXはワイルドカード仮想サーバーに割り当てられたIPアドレスを置き換えます。割り当てられたIPアドレスは127.0.0.1であり、これはDockerエンジンによってNetScaler CPXインスタンスに割り当てられたNSIPに置き換えられます。

高可用性NetScaler CPX展開では、プライマリNetScaler CPXインスタンスにワイルドカード仮想サーバーを追加できます。ノード間の構成同期により、セカンダリNetScaler CPXインスタンスにワイルドカード仮想サーバーが構成されます。これにより、DockerエンジンによってNetScaler CPXインスタンスに割り当てられたNSIP上で仮想サーバーを構成する必要がなくなります。

注意点:

- ワイルドカード仮想サーバーに割り当てるポート番号が、展開内の他の仮想サーバーによって使用されていないことを確認してください。
- ワイルドカード仮想サーバーに割り当てるポート番号が内部サービスによってすでに使用されている場合、ワイルドカード仮想サーバーの追加は失敗します。
- ワイルドカード仮想サーバーは「*」文字をサポートしていません。

ワイルドカードロードバランシング仮想サーバーを作成するには、コマンドプロンプトで次のコマンドを入力します。

```
add lb vserver <name> <serviceType> 127.0.0.1 <port>
```

```
add lb vserver testlbvserver HTTP 127.0.0.1 30000
```

ワイルドカードコンテンツスイッチング仮想サーバーを作成するには、コマンドプロンプトで次のコマンドを入力します。

```
add cs vserver <name> <serviceType> 127.0.0.1 <port>
```

```
add cs vserver testcsvserver HTTP 127.0.0.1 30000
```

プロキシとしてNetScaler CPXを展開して東西トラフィックフローを有効にする

この展開では、NetScaler CPXインスタンスはプロキシとして機能し、複数のホストに存在するアプリケーションコンテナ間の通信を可能にします。NetScaler CPXインスタンスは、複数のホスト内のアプリケーションとともにプロビジョニングされ、通信のための最短パスを提供します。

次の図は、NetScaler CPXインスタンスを介した2つのアプリケーション間のトラフィックフローを示しています。



この図は、アプリケーションCとアプリケーションBの間、およびアプリケーションAとアプリケーションBの間のトラフィックフローを示しています。アプリC（いずれかのホスト内）がBにリクエストを送信すると、そのリクエストはまず、アプリCと同じホスト上のNetScaler CPXコンテナで受信されます。その後、NetScaler CPXコンテナはトラフィックをアプリBと同じホストでホストされているNetScaler CPXコンテナに渡し、その後、トラフィックはアプリBに転送されます。アプリAがアプリBにリクエストを送信する場合も、同様のトラフィックパスがたどられます。

この例では、グローバルVIPを介してインターネットからアプリケーションへのトラフィックを許可するために、NetScaler MPXも展開されています。NetScaler MPXからのトラフィックはNetScaler CPXコンテナで受信され、その後、アプリケーションコンテナ全体にトラフィックを分散します。

次の図は、通信が行われるために設定する必要がある構成とともに、このトポロジを示しています。



次の表は、この例の構成でNetScaler CPXインスタンスに構成されているIPアドレスとポートをリストしています。



この例のシナリオを構成するには、3つのDockerホストすべてでNetScaler CPXコンテナを作成する際に、Linuxシェルスクリプトで次のコマンドを実行します。

```
docker run -dt -p 22 -p 80 -p 161/udp -p 30000-30002: 30000-30002 --ulimit core=-1 --privileged=true cpx:6.2
```

NetScaler Consoleのジョブ機能を使用するか、NITRO APIを使用して、次のコマンドを実行します。

Dockerホスト1上のNetScaler CPXインスタンスで:

```
add lb vserver VIP-A1 HTTP 172.17.0.2 30000

add service svc-A1 10.102.29.100 HTTP 80

bind lb vserver VIP-A1 svc-A1

add lb vserver VIP-B1 HTTP 172.17.0.2 30001

add service svc-B1 10.102.29.100 HTTP 90

bind lb vserver VIP-B1 svc-B1

add lb vserver VIP-C1 HTTP 172.17.0.2 30002

add service svc-VIP-C2 10.102.29.105 HTTP 30002

add service svc-VIP-C3 10.102.29.110 HTTP 30002

bind lb vserver VIP-C1 svc-VIP-C2

bind lb vserver VIP-C1 svc-VIP-C3
```

Dockerホスト2上のNetScaler CPXインスタンスで:

```
add lb vserver VIP-A2 HTTP 172.17.0.3 30000

add service svc-A2 10.102.29.105 HTTP 80

bind lb vserver VIP-A2 svc-A2

add lb vserver VIP-B2 HTTP 172.17.0.3 30001

add service svc-VIP-B1 10.102.29.100 HTTP 30001

bind lb vserver VIP-B2 svc-VIP-B1

add lb vserver VIP-C2 HTTP 172.17.0.3 30002
```

```
add service svc-C2 10.102.29.105 HTTP 70

bind lb vserver VIP-C2 svc-C2
```

Dockerホスト3上のNetScaler CPXインスタンスで:

```
add lb vserver VIP-A3 HTTP 172.17.0.4 30000

add service svc-VIP-A1 10.102.29.100 HTTP 30000

add service svc-VIP-A2 10.102.29.105 HTTP 30000

bind lb vserver VIP-A3 svc-VIP-A1

bind lb vserver VIP-A3 svc-VIP-A2

add lb vserver VIP-B3 HTTP 172.17.0.4 30001

add service svc-VIP-B1 10.102.29.100 HTTP 30001

bind lb vserver VIP-B3 svc-VIP-B1

add lb vserver VIP-C3 HTTP 172.17.0.4 30002

add service svc-C3 10.102.29.110 HTTP 70

bind lb vserver VIP-C3 svc-C3
```

シングルホストネットワークにNetScaler CPXを展開する

シングルホストネットワークでは、NetScaler CPXインスタンスは同じホスト上のアプリケーションコンテナ間のプロキシとして機能します。この役割において、NetScaler CPXインスタンスはコンテナベースのアプリケーションにスケーラビリティとセキュリティを提供します。さらに、パフォーマンスを最適化し、テレメトリデータへの洞察も提供します。

シングルホストネットワークでは、クライアント、サーバー、およびNetScaler CPXインスタンスは、同じDockerホスト上にコンテナとして展開されます。すべてのコンテナはdocker0ブリッジを介して接続されます。

この環境では、NetScaler CPXインスタンスは、同じDockerホスト上にコンテナとしてプロビジョニングされたアプリケーションのプロキシとして機能します。

次の図は、シングルホストトポロジを示しています。



この例では、Webアプリコンテナ (172.17.0.2) がクライアントであり、2つのデータベースコンテナ、DB1 (172.17.0.10) とDB2 (172.17.0.11) がサーバーです。NetScaler CPXコンテナ (172.17.0.4) は、クライアントとサーバーの間に位置し、プロキシとして機能します。

WebアプリケーションがNetScaler CPXを介してデータベースコンテナと通信できるようにするには、まず、2つのサーバーを表すためにNetScaler CPXコンテナ上で2つのサービスを設定する必要があります。次に、NetScaler CPXのIPアドレスと非標準のHTTPポート (81など) を使用して仮想サーバーを設定します。これは、NetScaler CPXがNITRO通信のために標準のHTTPポート80を予約しているためです。

このトポロジでは、クライアントとサーバーが同じネットワーク上にあるため、NATルールを設定する必要はありません。

このシナリオを設定するには、NetScaler ConsoleのJobs機能を使用するか、NITRO APIを使用して次のコマンドを実行します。

```
add service db1 HTTP 172.17.0.10 80
```

```
add service db2 HTTP 172.17.0.11 80
```

```
add lb vserver cpx-vip HTTP 172.17.0.4 81
```

```
bind lb vserver cpx-vip db1
```

```
bind lb vserver cpx-vip db2
```

マルチホストネットワークにNetScaler CPXを展開する

マルチホストネットワーク内のNetScaler CPXインスタンスは、ロードバランシング機能を提供するデータセンターの本番環境のデプロイメントで構成できます。さらに、監視機能と分析データも提供できます。

マルチホストネットワークでは、NetScaler CPXインスタンス、バックエンドサーバー、およびクライアントは異なるホストにデプロイされます。NetScaler CPXインスタンスがコンテナベースのアプリケーションとサーバーのセット、あるいは物理サーバーさえもロードバランシングする本番環境のデプロイメントで、マルチホストトポロジを使用できます。

トポロジ 1: NetScaler CPXとバックエンドサーバーが同じホスト上にある場合、クライアントが異なるネットワーク上にある場合

このトポロジでは、NetScaler CPXインスタンスとデータベースサーバーは同じDockerホストにプロビジョニングされますが、クライアントトラフィックはネットワーク上の別の場所から発信されます。このトポロジは、NetScaler CPXインスタンスがコンテナベースのアプリケーションまたはサーバーのセットをロードバランシングする本番環境のデプロイメントで使用される可能性があります。

次の図は、このトポロジを示しています。



この例では、NetScaler CPXインスタンス (172.17.0.4) と2つのサーバー、DB1 (172.17.0.10) およびDB2 (172.17.0.11) は、IPアドレス10.102.29.100を持つ同じDockerホストにプロビジョニングされます。クライアントはネットワーク上の別の場所にあります。

インターネットから発信されたクライアント要求は、NetScaler CPXインスタンスに構成されたVIPで受信され、その後、2つのサーバーに要求を分散します。

このトポロジを構成するには、次の2つの方法があります。

方法 1: VIPに別のIPアドレスと標準ポートを使用する

1. 追加のIPアドレスを使用して、NetScaler CPXコンテナでVIPを構成します。

2. Dockerホスト用に別のIPアドレスを構成します。
3. NATルールを構成して、Dockerホストの追加IPアドレスで受信したすべてのトラフィックをVIPの追加IPアドレスに転送します。
4. NetScaler CPXインスタンス上で2つのサーバーをサービスとして構成します。
5. 最後に、サービスをVIPにバインドします。

この構成例では、10.x.x.x ネットワークがパブリックネットワークを示していることに注意してください。

このシナリオ例を構成するには、NetScaler Console のジョブ機能を使用するか、NITRO API を使用して、次のコマンドを実行します。

```
add service s1 172.17.0.10 HTTP 80

add service s2 172.17.0.11 HTTP 80

add lb vserver cpx-vip HTTP 172.17.4.100 80

bind lb vserver cpx-vip s1

bind lb vserver cpx-vip s2
```

Docker ホスト用の追加のパブリック IP アドレスと NAT ルールを構成するには、Linux シェルプロンプトで次のコマンドを実行します。

```
ip addr add 10.102.29.103/24 dev eth0

iptables -t nat -A PREROUTING -p ip -d 10.102.29.103 -j DNAT --to-destination 172.17.4.100
```

方法 2: VIP に NetScaler CPX IP アドレスを使用し、ポートマッピングを構成する:

1. NetScaler CPX インスタンスで VIP と 2 つのサービスを構成します。VIP には非標準ポート 81 を使用します。
2. サービスをVIPにバインドします。
3. Docker ホストのポート 50000 で受信したすべてのトラフィックを VIP とポート 81 に転送する NAT ルールを構成します。

このシナリオ例を構成するには、3 つすべての Docker ホストで NetScaler CPX コンテナを作成する際に、Linux シェルプロンプトで次のコマンドを実行します。

```
docker run -dt -p 22 -p 80 -p 161/udp -p 50000:81 --ulimit core=-1 --privileged=true cpx:6.2
```

NetScaler CPX インスタンスがプロビジョニングされたら、NetScaler Console のジョブ機能を使用するか、NITRO API を使用して、次のコマンドを実行します。

```
add service s1 172.17.0.10 http 80

add service s2 172.17.0.11 http 80

add lb vserver cpx-vip HTTP 172.17.0.4 81

bind lb vserver cpx-vip s1

bind lb vserver cpx-vip s2
```

注:

NetScaler CPX インスタンスのプロビジョニング中にポートマッピングを構成していない場合は、Linux シェルプロンプトで次のコマンドを実行して NAT ルールを構成します。

```
iptables -t nat -A PREROUTING -p tcp -m addrtype --dst-type LOCAL -m tcp --dport 50000 -j DNAT --to-destination 172.17.0.4:81
```

トポロジ 2: 物理サーバーとクライアントを備えた NetScaler CPX

このトポロジでは、NetScaler CPX インスタンスのみが Docker ホストにプロビジョニングされます。クライアントとサーバーはコンテナベースではなく、ネットワーク上の別の場所に存在します。

この環境では、NetScaler CPX インスタンスを構成して、物理サーバー間でトラフィックを負荷分散できます。

次の図は、このトポロジを示しています。



この例では、NetScaler CPXコンテナ (172.17.0.4) がクライアントと物理サーバーの間にプロキシとして配置されています。サーバーDB1 (10.102.29.105) とDB2 (10.102.29.110) は、ネットワーク上のDockerホストの外部にあります。クライアントからのリクエストはインターネットから発信され、NetScaler CPXで受信され、2つのサーバーに分散されます。

クライアントとサーバー間のこの通信をNetScaler CPX経由で有効にするには、まずNetScaler CPXコンテナの作成時にポートマッピングを構成する必要があります。次に、2つのサーバーを表すようにNetScaler CPXコンテナ上で2つのサービスを構成します。最後に、NetScaler CPXのIPアドレスと非標準のマッピングされたHTTPポート8080を使用して仮想サーバーを構成します。

この構成例では、10.x.x.xネットワークはパブリックネットワークを示していることに注意してください。

このシナリオ例を構成するには、NetScaler CPXコンテナの作成時にLinuxシェルスクリプトで次のコマンドを実行します。

```
docker run -dt -p 22 -p 80 -p 161/udp -p 8080:8080 --ulimit core=-1 --privileged=true cpx:6.2
```

次に、NetScaler ConsoleのJobs機能を使用するか、NITRO APIを使用して、次のコマンドを実行します。

```
add service s1 HTTP 10.102.29.105 80

add service s2 HTTP 10.102.29.110 80

add lb vserver cpx-vip HTTP 172.17.0.4 8080

bind lb vserver cpx-vip s1

bind lb vserver cpx-vip s2
```

トポロジ3：異なるホストにプロビジョニングされたNetScaler CPXとサーバー

このトポロジでは、NetScaler CPXインスタンスとデータベースサーバーは異なるDockerホストにプロビジョニングされ、クライアントトラフィックはインターネットから発信されます。このト

ポロジは、NetScaler CPXインスタンスがコンテナベースのアプリケーションまたはサーバーのセットを負荷分散する本番環境の展開で使用される可能性があります。

次の図は、このトポロジを示しています。



この例では、NetScaler CPXインスタンスとサーバー (DB1) は、IPアドレス10.102.29.100を持つ同じDockerホストにプロビジョニングされています。他の4つのサーバー (DB2、DB3、DB4、DB5) は、10.102.29.105と10.102.29.110の2つの異なるDockerホストにプロビジョニングされています。

インターネットから発信されたクライアントリクエストはNetScaler CPXインスタンスで受信され、その後、5つのサーバーにリクエストを分散します。この通信を有効にするには、以下を構成する必要があります。

1. NetScaler CPXコンテナの作成時にポートマッピングを設定します。この例では、コンテナ上のポート8080をホスト上のポート8080に転送する必要があることを意味します。クライアントリクエストがホストのポート8080に到着すると、CPXコンテナのポート8080にマッピングされます。
2. NetScaler CPXインスタンス上で5つのサーバーをサービスとして構成します。これらのサービスを設定するには、それぞれのDockerホストのIPアドレスとマッピングされたポートの組み合わせを使用する必要があります。
3. クライアント要求を受信するために、NetScaler CPXインスタンス上にVIPを構成します。このVIPは、NetScaler CPXのIPアドレスと、ホストのポート8080にマッピングされたポート8080で表される必要があります。
4. 最後に、サービスをVIPにバインドします。

この構成例では、10.x.x.xネットワークはパブリックネットワークを示していることに注意してください。

このシナリオ例を構成するには、NetScaler CPXコンテナの作成中に、Linuxシェルプロンプトで次のコマンドを実行します。

```
docker run -dt -p 22 -p 80 -p 161/udp -p 8080:8080 --ulimit core=-1 --privileged=true cpx:6.2
```

NetScaler Consoleのジョブ機能を使用するか、NITRO APIを使用して、次のコマンドを実行します。

```
add service s1 10.102.29.100 HTTP 8081

add service s2 10.102.29.105 HTTP 8081

add service s3 10.102.29.105 HTTP 8082

add service s4 10.102.29.110 HTTP 8081

add service s5 10.102.29.110 HTTP 8082

add lb vserver cpx-vip HTTP 172.17.0.2 8080

bind lb vserver cpx-vip s1

bind lb vserver cpx-vip s2

bind lb vserver cpx-vip s3

bind lb vserver cpx-vip s4

bind lb vserver cpx-vip s5
```

ネットワークに直接アクセスして NetScaler CPXを展開する

ブリッジネットワークモードでは、NetScaler CPXインスタンスがネットワークに直接アクセスできるように構成できます。このシナリオでは、受信トラフィックはNetScaler CPX仮想サーバーIP (VIP) で直接受信されます。

この通信を有効にするには、まずdocker0ブリッジにパブリックIPアドレスを構成する必要があります。次に、ネットワークポートeth0からパブリックIPアドレスを削除し、ネットワークポートをdocker0ブリッジにバインドします。

2つのサービスを追加して負荷分散を構成し、次にネットワークパブリックIPアドレスをNetScaler CPXインスタンスのVIPとして構成します。クライアント要求はVIPで直接受信されます。

構成例では、10.x.x.xネットワークはパブリックネットワークを示します。

このシナリオを構成するには、Linuxシェルスクリプトで次のコマンドを実行します。

```
ip addr add 10.102.29.100/24 dev docker0;

ip addr del 10.102.29.100/24 dev eth0;

brctl addif docker0 eth0;

ip route del default;

ip route add default via 10.102.29.1 dev docker0
```

NetScaler ConsoleのJobs機能を使用するか、NITRO APIを使用して、次のコマンドを実行します。

```
add service s1 172.17.0.8 http 80

add service s2 172.17.0.9 http 80

add lb vserver cpx-vip HTTP 10.102.29.102 80

bind lb vserver cpx-vip s1
```

```
bind lb vserver cpx-vip s2
```

ConfigMapsを使用してKubernetesでNetScaler CPXを構成する

Kubernetesでは、ConfigMapsを使用してNetScaler CPXインスタンスを構成できます。

ConfigMapsを使用すると、インスタンスの起動中にNetScaler CPXインスタンスを動的に構成できます。

NetScaler CPXインスタンスで動的に実行したいNetScaler固有の構成とbashシェルコマンドを含む `cpx.conf` 構成ファイルを作成します。構成ファイルの構造には、`#NetScaler Commands` と `#Shell Commands` の2種類のタグが必要です。`#NetScaler Commands` タグの下には、NetScaler CPXインスタンスでNetScaler固有の構成を設定するためのすべてのNetScalerコマンドを追加する必要があります。`#Shell Commands` タグの下には、NetScaler CPXインスタンスで実行したいシェルコマンドを追加する必要があります。

重要:

- タグは構成ファイル内で複数回繰り返すことができます。
- 構成ファイルにはコメントを含めることもできます。コメントの前に「#」文字を追加してください。
- タグは大文字と小文字を区別しません。
- 構成ファイルを使用してNetScaler CPXコンテナを展開する際に障害が発生した場合、その障害は `ns.log` ファイルに記録されます。
- NetScaler CPXインスタンスの起動後、ConfigMapを変更した場合、更新された構成はNetScaler CPXインスタンスが再起動されたときにのみ適用されます。

以下は構成ファイルのサンプルです。

```
#NetScaler Commands

add lb vserver v1 http 1.1.1.1 80

add service s1 2.2.2.2 http 80

bind lb vserver v1 s1
```

```
#Shell Commands

touch /etc/a.txt

echo "this is a" > /etc/a.txt

#NetScaler Commands

add lb vserver v2 http

#Shell Commands

echo "this is a 1" >> /etc/a.txt

#NetScaler Commands

add lb vserver v3 http
```

構成ファイルを作成したら、`kubectl create configmap` コマンドを使用して構成ファイルからConfigMapを作成する必要があります。

```
kubectl create configmap cpx-config --from-file=cpx.conf
```

上記の例では、構成ファイルcpx.confに基づいてConfigMap `cpx-config` を作成できます。その後、このConfigMapをNetScaler CPXインスタンスの展開に使用するYAMLファイルで使用できます。

`kubectl get configmap` コマンドを使用して、作成されたConfigMapを表示できます。

```
root@node1:~/yaml# kubectl get configmap cpx-config -o yaml
```

サンプル:

```
apiVersion: v1

data:

  cpx.conf: |

    #NetScaler Commands

    add lb vserver v1 http 1.1.1.1 80
```

```
    add service s1 2.2.2.2 http 80

    bind lb vserver v1 s1

#Shell Commands

    touch /etc/a.txt

    echo "this is a" > /etc/a.txt

    echo "this is the file" >> /etc/a.txt

    ls >> /etc/a.txt

#NetScaler Commands

    add lb vserver v2 http

#Shell Commands

    echo "this is a 1" >> /etc/a.txt

#NetScaler Commands

    add lb vserver v3 http

#end of file

kind: ConfigMap

metadata:

    creationTimestamp: 2017-12-26T06:26:50Z

    name: cpx-config

    namespace: default

    resourceVersion: "8865149"

    selfLink: /api/v1/namespaces/default/configmaps/cpx-config
```

```
uid: c1c7cb5b-ea05-11e7-914a-926745c10b02
```

作成したConfigMap `cpx-config` を、NetScaler CPXインスタンスの展開に使用するYAMLファイルで次のように指定できます。

```
apiVersion: v1

kind: Pod

metadata:

  name: cpx-1

  labels:

    app: cpx-daemon

  annotations:

    NETSCALER_AS_APP: "True"

spec:

  hostNetwork: true

  containers:

  - name: cpx

    image: "quay.io/citrix/citrix-k8s-cpx-ingress:13.1-33.47"

    securityContext:

      privileged: true

    volumeMounts:

      - name: config-volume

        mountPath: /cpx/bootup_conf

    env:
```

```
- name: "EULA"

  value: "yes"

- name: "NS_NETMODE"

  value: "HOST"

- name: "kubernetes_url"

  value: "https://10.90.248.101:6443"

- name: "NS_MGMT_SERVER"

  value: "10.90.248.99"

- name: "NS_MGMT_FINGER_PRINT"

  value:
"19:71:A3:36:85:0A:2B:62:24:65:0F:7E:72:CC:DC:AD:B8:BF:53:1E"

- name: "NS_ROUTABLE"

  value: "FALSE"

- name: "KUBERNETES_TASK_ID"

  valueFrom:

    fieldRef:

      fieldPath: metadata.name

  imagePullPolicy: Never

volumes:

- name: config-volume

  configMap:
```

```
name: cpx-config
```

NetScaler CPXインスタンスがデプロイされ、ConfigMapで指定された構成が開始されると、`cpx-config` がNetScaler CPXインスタンスに適用されます。

KubernetesノードのローカルDNSキャッシュとしてNetScaler CPXをデプロイする

Kubernetesクラスター内のアプリケーションポッドは、他のアプリケーションポッドと通信するためにDNSに依存しています。Kubernetesクラスター内のアプリケーションからのDNSリクエストは、Kubernetes DNS (kube-dns) によって処理されます。マイクロサービスアーキテクチャの普及により、Kubernetesクラスター内のDNSリクエストレートが増加しています。その結果、Kubernetes DNS (kube-dns) は過負荷になっています。これにより、各KubernetesノードにNetScaler CPXをローカルDNSキャッシュとしてデプロイし、ノード内のアプリケーションポッドからのDNSリクエストをNetScaler CPXに転送できます。したがって、DNSリクエストをより迅速に解決し、Kubernetes DNSの負荷を大幅に軽減できます。

NetScaler CPXをデプロイするには、Kubernetes DaemonSetエンティティを使用して、Kubernetesクラスター内の各ノードにNetScaler CPXポッドをスケジュールします。Kubernetes DaemonSetは、クラスター内の各KubernetesノードにNetScaler CPXのインスタンスが存在することを保証します。アプリケーションポッドがCPX DNSポッドにトラフィックを直接送信するようにするには、エンドポイントとしてNetScaler CPXポッドを持つKubernetesサービスを作成する必要があります。このサービスのクラスターIPは、アプリケーションポッドのDNSエンドポイントとして使用されます。アプリケーションポッドがDNS解決のためにNetScaler CPXサービスクラスターIPアドレスを使用するようにするには、各ノードのkubelet構成ファイルをNetScaler CPXサービスクラスターIPで更新する必要があります。

NodeLocal DNSキャッシュとしてNetScaler CPXのデプロイをサポートするために、次の環境変数が導入されています。

- **KUBE_DNS_SVC_IP** : **kube-dns** サービスのクラスターIPアドレスを指定します。これは、NetScaler CPXポッドでの構成をトリガーするための必須引数です。NetScaler CPXキャッシュにDNSクエリ応答がない場合、NetScaler CPXポッドはこのIPアドレスにDNSクエリを送ります。
- **CPX_DNS_SVC_IP** : NetScaler CPXサービスのクラスターIPアドレスを指定します。
CPX_DNS_SVC_IP 環境変数は、ノード上のローカルDNSを構成するために使用されます。この変数を構成すると、アプリケーションポッドから発信されるDNSリクエストをノード内のローカルNetScaler CPXポッドに転送するための **iptables** ルールが追加されます。

- `NS_DNS_FORCE_TCP` : この環境変数は、クエリがUDP経由で受信された場合でも、DNSリクエストにTCPの使用を強制します。
- `NS_DNS_EXT_RESLV_IP` : 特定のドメインのDNSリクエストを転送するために、外部ネームサーバーのIPアドレスを指定します。
- `NS_DNS_MATCH_DOMAIN` : クエリを外部ネームサーバーに転送するために、照合する外部ドメイン文字列を指定します。

ノード上にNetScaler CPXをDNSキャッシュとしてデプロイする

KubernetesクラスターのローカルDNSキャッシュとしてNetScaler CPXをデプロイすることには、次のタスクが含まれます。

マスターノードで:

- エンドポイントとしてNetScaler CPXポッドを持つKubernetesサービスを作成する
- NetScaler CPXポッドの環境変数を定義するためのConfigMapを作成する
- Kubernetes DaemonSetを使用して、Kubernetesクラスター内の各ノードにNetScaler CPXポッドをスケジュールします。

ワーカーノードで:

- kubelet 設定ファイルを NetScaler CPX サービスのクラスター IP アドレスで変更し、DNS リクエストを NetScaler CPX に転送します。

Kubernetes マスターノードでの設定

ノードのローカル DNS キャッシュとして NetScaler CPX をデプロイするには、Kubernetes マスターノードで次の手順を実行します。

1. `cpx_dns_svc.yaml` ファイルを使用して、NetScaler CPX ポッドをエンドポイントとするサービスを作成します。

```
kubectl apply -f cpx_dns_svc.yaml
```

`cpx_dns_svc.yaml` ファイルは次のとおりです。

```
apiVersion: v1

kind: Service

metadata:

  name: cpx-dns-svc

  labels:

    app: cpxd

spec:

  ports:

    - protocol: UDP

      port: 53

      name: dns

    - protocol: TCP

      port: 53

      name: dns-tcp

  selector:

    app: cpx-daemon
```

2. NetScaler CPX サービスの IP アドレスを取得します。

```
kubectl get svc cpx-dns-svc
```

3. Kube DNS サービスの IP アドレスを取得します。

```
kubectl get svc -n kube-system
```

4. NetScaler CPX ポッドの環境変数を定義する ConfigMap を作成します。これらの環境変数は、NetScaler CPX サービスと Kube DNS サービスの IP アドレスを渡すために使用されます。この手順では、ファイル内でデータ (キーと値のペア) として指定された環境変数を使用して、サンプル ConfigMap `cpx-dns-cache` を作成します。

```
kubectl create configmap cpx-dns-cache --from-file <path-to-file>
```

環境変数をキーと値のペアとして含むサンプルファイルは次のとおりです。

```
CPX_DNS_SVC_IP: 10.111.95.145

EULA: "yes"

KUBE_DNS_SVC_IP: 10.96.0.10

NS_CPX_LITE: "1"

NS_DNS_EXT_RESLV_IP: 10.102.217.142

NS_DNS_MATCH_DOMAIN: citrix.com

PLATFORM: CP1000
```

サンプル ConfigMap は次のとおりです。

```
apiVersion: v1

data:

  CPX_DNS_SVC_IP: 10.111.95.145

  EULA: "yes"

  KUBE_DNS_SVC_IP: 10.96.0.10

  NS_CPX_LITE: "1"

  NS_DNS_EXT_RESLV_IP: 10.102.217.142

  NS_DNS_MATCH_DOMAIN: citrix.com
```

```
PLATFORM: CP1000

kind: ConfigMap

metadata:

  creationTimestamp: "2019-10-15T07:45:54Z"

  name: cpx-dns-cache

  namespace: default

  resourceVersion: "8026537"

  selfLink: /api/v1/namespaces/default/configmaps/cpx-dns-cache

  uid: 8d06f6ee-133b-4e1a-913c-9963cbf4f48
```

5. マスターノードに NetScaler CPX 用の Kubernetes DaemonSet を作成します。

```
kubectl apply -f cpx_daemonset.yaml
```

`cpx_daemonset.yaml` ファイルは次のとおりです。

```
apiVersion: apps/v1

kind: DaemonSet

metadata:

  name: cpx-daemon

  labels:

    app: cpxd

spec:

  selector:

    matchLabels:
```

```
        app: cpx-daemon

template:

  metadata:

    labels:

      app: cpx-daemon

  spec:

    containers:

      - name: cpxd

        imagePullPolicy: IfNotPresent

        image: localhost:5000/dev/cpx

        volumeMounts:

          - mountPath: /netns/default/

            name: test-vol

        ports:

          - containerPort: 53

    envFrom:

      - configMapRef:

          name: cpx-dns-cache

    securityContext:

      privileged: true

      allowPrivilegeEscalation: true

    capabilities:
```

```

    add: ["NET_ADMIN"]

volumes:

  - name: test-vol

    hostPath:

      path: /proc/1/ns

      type: Directory

```

Kubernetes クラスター内のワーカーノードでの設定

マスターノードでの設定が完了したら、ワーカーノードで次の手順を実行します。

1. アプリケーションポッドがDNS解決にNetScaler CPXサービスクラスターIPを使用できるように、以下のいずれかの手順を使用してkubelet設定ファイルを変更します。

- ノードのkubeletを再構成するの順に従い、`--cluster-dns` 引数の値を以下の形式で変更します。

```
--cluster-dns=&lt;CPX_DNS_SVC_IP>,&lt;KUBE_DNS_SVC_IP>
```

または

- `/etc/systemd/system/kubelet.service.d/10-kubeadm.conf` ファイルを編集し、以下の手順を使用して `--cluster-dns` 引数を変更します。
 - a. kubelet設定を編集し、NetScaler CPXサービスのクラスターIPアドレスと `kube-dns` サービスIPアドレスを `--cluster-dns` 引数に指定します。

```

root@node:~# cat /etc/systemd/system/kubelet.service.d/10-
kubeadm.conf | grep KUBELET_DNS_ARGS

Environment="KUBELET_DNS_ARGS=--cluster-
dns=10.111.95.145,10.96.0.10 --cluster-domain=cluster.local"

```

```
ExecStart=/usr/bin/kubelet $KUBELET_KUBECONFIG_ARGS  
$KUBELET_CONFIG_ARGS $KUBELET_DNS_ARGS
```

- b. 以下のコマンドを使用してノードのkubeletをリロードします。

```
# systemctl daemon-reload  
  
# service kubelet restart
```

Google Compute Engine に NetScaler CPX プロキシを展開する

この展開ガイドでは、エンタープライズネットワーク内で実行されているNetScaler Consoleを使用して、Google CloudのGoogle Compute Engine (GCE) にDockerでNetScaler CPXを展開する方法について説明します。この展開では、GCEにインストールされたNetScaler CPXが2つのバックエンドサーバーの負荷分散を行い、NetScaler Consoleがライセンスおよび分析ソリューションを提供します。

NetScaler CPXは、完全なレイヤー7機能、SSLオフロード、複数のプロトコル、およびNITRO APIをサポートするコンテナベースのプロキシです。NetScaler Consoleは、管理、ライセンス、および分析ソリューションを提供します。ライセンスサーバーとして、NetScaler Consoleは、オンプレミスまたはクラウドで実行されるNetScaler CPXインスタンスにエンタイトルメントを提供します。

CPXとCPX Expressは同じイメージです。NetScaler Consoleを使用してCPXイメージをライセンス供与およびインストールすると、Docker App StoreのCPXイメージ（リリース11または12）は完全なCPXインスタンスになります。ライセンスがない場合、CPXイメージは20 Mbpsと250のSSL接続をサポートするCPX Expressインスタンスになります。

前提条件

- NetScaler CPX専用の2 GBメモリと1 vCPU
- GCEから入手可能なDockerオープンソース
- GCEへのインターネットまたはVPN接続を備えたオンプレミスで実行されているNetScaler Console

注

NetScaler Consoleの展開方法については、「[NetScaler Consoleの展開](#)」を参照してください。

設定手順

この展開を構成するには、次の手順を実行する必要があります。

1. GCE VMにDockerをインストールします。
2. DockerインスタンスとのリモートAPI通信を構成します。
3. NetScaler CPXイメージをインストールします。
4. CPXインスタンスを作成します。
5. NetScaler Console を介して NetScaler CPX のライセンスを付与します。
6. NetScaler CPX で負荷分散サービスを設定し、構成を確認します。
 - a. NGINX Webサーバーをインストールします。
 - b. 負荷分散のために NetScaler CPX を構成し、両方のWebサービスへの負荷分散を確認します。

ステップ1：GCE VMにDockerをインストールする

GCEからLinux Ubuntu VMを作成します。次に、次の例に示すコマンドを使用して、VMにDockerをインストールします。

```
$ sudo curl -ssl https://get.docker.com/ | sh

% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed

0 0 0 0 0 0 0 0 --:--:-- --:--:-- --:--:-- 0curl: (6) Could not resolve
host: xn--ssl-1n0a

100 17409 100 17409 0 0 21510 0 --:--:-- --:--:-- --:--:-- 21492

apparmor is enabled in the kernel and apparmor utils were already
installed

\+ sudo -E sh -c apt-key add -

\+ echo -----BEGIN PGP PUBLIC KEY BLOCK-----

Version: GnuPG v1

mQINBFWln24BEADrBl5p99uKh8+rpvqJ48u4eTtjeXAWbslJotmC/CakbNSqOb9o
```

ddfzRvGVeJVERt/Q/m1vEqgnyTQy+e6oEYN2Y2kqXceUhXagThnqCoxcEJ3+KM4R
mYdoe/BJ/J/6rH0jq70mk24z2qB3RU1uAv57iY5VGw5p45uZB4C4pNNsBJXoCvPn
TGAs/7IrekFZDDgVraPx/hdiwopQ8NltSfZCyu/jPpWFK28TR8yfVlZyFwibj5WK
dHM7ZTq1A1tHIG+agyPf3Rae0jPMsHR6q+arXVwMccy0i+ULU0z8mHUJ3iEMIrpt
X+80KaN/ZjibfsB0CjcfiJSB/acn4nxQQgNZigna32velafhQivsNREFeJpzENiG
H0oyC6qVe0gKrRiKxzymj0FIMLru/iFF5pSwcBQB7PYlt8J0G80lAcPr6VCiN+4c
NKv03SdvA69dC0j79Pu09IivQsJXsSq96HB+TeEmmL+xSdpGtGdCJHHM1fDeCqkZ
hT+RtBGQL2SEdWjxbF43oQopocT8chvyX6Za1tn0svoGs+wX3Z/H6/8P5anog43U
65c0A+64Jj00rNDR8j31izhtQMRo892kGeQAaaxg4Pz6HnS7hRC+c0MHUU4HA7iM
zHrouAdYeTZeZEQ0A7SxtCME9ZnGwe2grxPXh/U/80WJGkzLFNcTKdv+rWARAQAB
tDdEb2NrZXIgUmVsZWfzZSBub29sIChyZwx1YXNlZG9ja2VyKSA8ZG9ja2VyQGRv
Y2t1ci5jb20+iQIcBBABCgAGBQJWw7vdAAoJEFyzYeVS+w0QHysP/i37m4Syo0CV
cnybl18vzwBEcp4VCRbXvHv0Xty1gccVIV8/aJqNKgBV97lY3vrp0yiIeB8ETQeg
srxFE7t/Gz0rsL0bqfLEHdmn5iBJRkhLfCpzje0nyB3Z0IJB6Uog0/msQVYe5CXJ
l6uwr0AmoiCBLrVlDAktxVh9RWch0l0KZRX2FpHu8h+uM0/zySqIidlYfLa3y5oH
scU+nGU1i6ImwDTD3ysZC5jp9aVfvUmcESyAb4vvdCAHR+bXhA/RW8QHeeMFliww
7Z2jYHyuHmDnWG2yUrnCqAJTrWV+0fKRIzzJFBs4e88ru5h2ZIXdRepw/+C0Yj34
LyzxR2cxr2u/xvxwXCkSMe7F4KZAphD+1ws61FhnUMi/PERMYfTFuvPrCkq4gyBj
t3fFpZ2NR/fKW87Q0eVcn1ivXl9id3MMs9KXJsg7QasT7mCsee2VIFsrxrkFQ2jNp
D+JAERRn9Fj4ArHL5TbwkkFbZZvSi6fr5h2GbCAXIGhIXKnjjorPY/YDX6X8AaH0
W1zb1Wy/CFr6VF1963jrjJgag0G6tNtBZLrc1ZgWh0QpeZZ5Lbvz2ZA5CqRrFAVc
wPNW1f0bFIRtqV6vuVluFOPCMAAnOnqR02w9t17iVQj03oVN0mbQi9vjuExXh1Yo

```
ScVeti06LSm1QfVEVRTqHLMgXyR/EMo7iQIcBBABCgAGBQJXSWBIAAoJEFyzYeVS
```

```
\+w0QeH0QAI6btAfYwYPuAjfRUy9q1nPhZ+xt1rnwsUzsbmo8K3XTNh+1/R08nu0d
```

```
sczw30Q1wju28fh1N8ay223+69f0+yICaXqR18AbGgFGKX7vo0gfEVaxdItUN3eH
```

```
NydGFzme0KbAlrxIMECnSTG/TkFVY09Nt1v9vSN2BupmTagTRExLZKnVsWRzp+X
```

```
\-----END PGP PUBLIC KEY BLOCK-----
```

```
OK
```

```
\+ sudo -E sh -c mkdir -p /etc/apt/sources.list.d
```

```
\+ dpkg --print-architecture
```

```
\+ sudo -E sh -c echo deb \\\[arch=amd64\\\] https://  
apt.dockerproject.org/repo ubuntu-yakkety main \> /etc/apt/sources.list.d/  
docker.list
```

```
\+ sudo -E sh -c sleep 3; apt-get update; apt-get install -y -q docker-  
engine
```

```
Hit:1 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety InRelease
```

```
Get:2 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates  
InRelease \[102 kB\]
```

```
Get:3 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports  
InRelease \[102 kB\]
```

```
Get:4 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/restricted  
Sources \[5,376 B\]
```

```
Get:5 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/multiverse  
Sources \[181 kB\]
```

```
Get:6 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/universe  
Sources \[8,044 kB\]
```

```
Get:7 http://archive.canonical.com/ubuntu yakkety InRelease \[11.5 kB\]

Get:8 http://security.ubuntu.com/ubuntu yakkety-security InRelease \[102
kB\]

Get:9 https://apt.dockerproject.org/repo ubuntu-yakkety InRelease \[47.3
kB\]

Get:10 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/main Sources
\[903 kB\]

Get:11 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
restricted Sources \[2,688 B\]

Get:12 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
universe Sources \[57.9 kB\]

Get:13 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
multiverse Sources \[3,172 B\]

Get:14 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/main
Sources \[107 kB\]

Get:15 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/main
amd64 Packages \[268 kB\]

Get:16 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/main
Translation-en \[122 kB\]

Get:17 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
universe amd64 Packages \[164 kB\]

Get:18 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
universe Translation-en \[92.4 kB\]

Get:19 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
multiverse amd64 Packages \[4,840 B\]

Get:20 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
multiverse Translation-en \[2,708 B\]

Get:21 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports/
universe Sources \[2,468 B\]
```

```
Get:22 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports/  
main Sources \[2,480 B\  
  
Get:23 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports/  
main amd64 Packages \[3,500 B\  
  
Get:24 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports/  
universe amd64 Packages \[3,820 B\  
  
Get:25 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports/  
universe Translation-en \[1,592 B\  
  
Get:26 http://archive.canonical.com/ubuntu yakkety/partner amd64 Packages  
\[2,480 B\  
  
Get:27 http://security.ubuntu.com/ubuntu yakkety-security/main Sources  
\[47.7 kB\  
  
Get:28 https://apt.dockerproject.org/repo ubuntu-yakkety/main amd64  
Packages \[2,453 B\  
  
Get:29 http://security.ubuntu.com/ubuntu yakkety-security/universe Sources  
\[20.7 kB\  
  
Get:30 http://security.ubuntu.com/ubuntu yakkety-security/multiverse  
Sources \[1,140 B\  
  
Get:31 http://security.ubuntu.com/ubuntu yakkety-security/restricted  
Sources \[2,292 B\  
  
Get:32 http://security.ubuntu.com/ubuntu yakkety-security/main amd64  
Packages \[150 kB\  
  
Get:33 http://security.ubuntu.com/ubuntu yakkety-security/main  
Translation-en \[68.0 kB\  
  
Get:34 http://security.ubuntu.com/ubuntu yakkety-security/universe amd64  
Packages \[77.2 kB\  
  
Get:35 http://security.ubuntu.com/ubuntu yakkety-security/universe  
Translation-en \[47.3 kB\  
  
Get:36 http://security.ubuntu.com/ubuntu yakkety-security/multiverse amd64
```

```
Packages \[2,832 B\]

Fetched 10.8 MB in 2s (4,206 kB/s)

Reading package lists... Done

Reading package lists...

Building dependency tree...

Reading state information...

The following additional packages will be installed:

aufs-tools cgroupfs-mount libltdl7

The following NEW packages will be installed:

aufs-tools cgroupfs-mount docker-engine libltdl7

0 upgraded, 4 newly installed, 0 to remove and 37 not upgraded.

Need to get 21.2 MB of archives.

After this operation, 111 MB of additional disk space will be used.

Get:1 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/universe amd64
aufs-tools amd64 1:3.2+20130722-1.1ubuntu1 \[92.9 kB\]

Get:2 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/universe amd64
cgroupfs-mount all 1.3 \[5,778 B\]

Get:3 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/main amd64
libltdl7 amd64 2.4.6-1 \[38.6 kB\]

Get:4 https://apt.dockerproject.org/repo ubuntu-yakkety/main amd64 docker-
engine amd64 17.05.0~ce-0~ubuntu-yakkety \[21.1 MB\]

Fetched 21.2 MB in 1s (19.8 MB/s)

Selecting previously unselected package aufs-tools.

(Reading database ... 63593 files and directories currently installed.)
```

```
Preparing to unpack .../aufs-  
tools\_1%3a3.2+20130722-1.1ubuntu1\_amd64.deb ...  
  
Unpacking aufs-tools (1:3.2+20130722-1.1ubuntu1) ...  
  
Selecting previously unselected package cgroupfs-mount.  
  
Preparing to unpack .../cgroupfs-mount\_1.3\_all.deb ...  
  
Unpacking cgroupfs-mount (1.3) ...  
  
Selecting previously unselected package libltdl7:amd64.  
  
Preparing to unpack .../libltdl7\_2.4.6-1\_amd64.deb ...  
  
Unpacking libltdl7:amd64 (2.4.6-1) ...  
  
Selecting previously unselected package docker-engine.  
  
Preparing to unpack .../docker-engine\_17.05.0~ce-0~ubuntu-  
yakkety\_amd64.deb ...  
  
Unpacking docker-engine (17.05.0~ce-0~ubuntu-yakkety) ...  
  
Setting up aufs-tools (1:3.2+20130722-1.1ubuntu1) ...  
  
Processing triggers for ureadahead (0.100.0-19) ...  
  
Setting up cgroupfs-mount (1.3) ...  
  
Processing triggers for libc-bin (2.24-3ubuntu2) ...  
  
Processing triggers for systemd (231-9ubuntu4) ...  
  
Setting up libltdl7:amd64 (2.4.6-1) ...  
  
Processing triggers for man-db (2.7.5-1) ...  
  
Setting up docker-engine (17.05.0~ce-0~ubuntu-yakkety) ...  
  
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service  
→ /lib/systemd/system/docker.service.
```

```
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /lib/systemd/system/docker.socket.
```

```
Processing triggers for ureadahead (0.100.0-19) ...
```

```
Processing triggers for libc-bin (2.24-3ubuntu2) ...
```

```
Processing triggers for systemd (231-9ubuntu4) ...
```

```
\+ sudo -E sh -c docker version
```

```
Client:
```

```
Version: 17.05.0-ce
```

```
API version: 1.29
```

```
Go version: go1.7.5
```

```
Git commit: 89658be
```

```
Built: Thu May 4 22:15:36 2017
```

```
OS/Arch: linux/amd64
```

```
Server:
```

```
Version: 17.05.0-ce
```

```
API version: 1.29 (minimum version 1.12)
```

```
Go version: go1.7.5
```

```
Git commit: 89658be
```

```
Built: Thu May 4 22:15:36 2017
```

```
OS/Arch: linux/amd64
```

```
Experimental: false
```

If you would like to use Docker as a non-root user, you should now consider

adding your user to the "docker" group with something like:

```
sudo usermod -aG docker albert\_lee
```

Remember that you will have to log out and back in for this to take effect.

WARNING: Adding a user to the "docker" group will grant the ability to run containers which can be used to obtain root privileges on the docker host.

Refer to <https://docs.docker.com/engine/security/security/#docker-daemon-attack-surface>

for more information.

\$

```
**$ sudo docker info**
```

```
Containers: 0
```

```
Running: 0
```

```
Paused: 0
```

```
Stopped: 0
```

```
Images: 0

Server Version: 17.05.0-ce

Storage Driver: aufs

Root Dir: /var/lib/docker/aufs

Backing Filesystem: extfs

Dirs: 0

Dirperm1 Supported: true

Logging Driver: json-file

Cgroup Driver: cgroupfs

Plugins:

Volume: local

Network: bridge host macvlan null overlay

Swarm: inactive

Runtimes: runc

Default Runtime: runc

Init Binary: docker-init

containerd version: 9048e5e50717ea4497b757314bad98ea3763c145

runc version: 9c2d8d184e5da67c95d601382adf14862e4f2228

init version: 949e6fa

Security Options:

apparmor

seccomp
```

```
Profile: default

Kernel Version: 4.8.0-51-generic

Operating System: Ubuntu 16.10

OSType: linux

Architecture: x86\_64

CPUs: 1

Total Memory: 3.613GiB

Name: docker-7

ID: R5TW:VKXK:EKGR:GHWM:UNU4:LPJH:IQY5:X77G:NNRQ:HWBY:LIUD:4ELQ

Docker Root Dir: /var/lib/docker

Debug Mode (client): false

Debug Mode (server): false

Registry: https://index.docker.io/v1/

Experimental: false

Insecure Registries:

127.0.0.0/8

Live Restore Enabled: false

WARNING: No swap limit support

$

**$ sudo docker images**
```

```
REPOSITORY TAG IMAGE ID CREATED SIZE
```

```
$
```

```
**$ sudo docker ps**
```

```
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
```

```
$
```

ステップ2：DockerインスタンスとのリモートAPI通信を構成する

DockerインスタンスとのAPI通信のためにポート4243を開きます。このポートは、NetScaler ConsoleがDockerインスタンスと通信するために必要です。

```
**cd /etc/systemd/system**

**sudo vi docker-tcp.socket**

**cat docker-tcp.socket**

[Unit\

**Description=Docker Socket for the API

\[Socket\

ListenStream=4243

BindIPv6only=both

Service=docker.service

\[Install\

WantedBy=sockets.target**
```

```
$ **sudo systemctl enable docker-tcp.socket**
```

```
Created symlink /etc/systemd/system/sockets.target.wants/docker-tcp.socket  
→ /etc/systemd/system/docker-tcp.socket.
```

```
**sudo systemctl enable docker.socket**
```

```
**sudo systemctl stop docker**
```

```
**sudo systemctl start docker-tcp.socket**
```

```
**sudo systemctl start docker**
```

```
$ **sudo systemctl status docker**
```

- `docker.service` - Docker Application Container Engine

```
Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor  
preset: enabled)
```

```
Active: **active (running)** since Wed 2017-05-31 12:52:17 UTC; 2s ago
```

```
Docs: https://docs.docker.com
```

```
Main PID: 4133 (dockerd)
```

```
Tasks: 16 (limit: 4915)
```

```
Memory: 30.1M
```

```
CPU: 184ms
```

```
CGroup: /system.slice/docker.service
```

```
└─4133 /usr/bin/dockerd -H fd://
```

```
└─4137 docker-containerd -l unix:///var/run/docker/libcontainerd/docker-  
containerd.sock --metrics-interval=0 --start-timeout 2m -
```

```
May 31 12:52:17 docker-7 dockerd\[4133\]:
time="2017-05-31T12:52:17.300890402Z" level=warning msg="Your kernel does
not support cgroup rt peri

May 31 12:52:17 docker-7 dockerd\[4133\]:
time="2017-05-31T12:52:17.301079754Z" level=warning msg="Your kernel does
not support cgroup rt runt

May 31 12:52:17 docker-7 dockerd\[4133\]:
time="2017-05-31T12:52:17.301681794Z" level=info msg="Loading containers:
start."

May 31 12:52:17 docker-7 dockerd\[4133\]:
time="2017-05-31T12:52:17.417539064Z" level=info msg="Default bridge
(docker0) is assigned with an I

May 31 12:52:17 docker-7 dockerd\[4133\]:
time="2017-05-31T12:52:17.465011600Z" level=info msg="Loading containers:
done."

May 31 12:52:17 docker-7 dockerd\[4133\]:
time="2017-05-31T12:52:17.484747909Z" level=info msg="Daemon has completed
initialization"

May 31 12:52:17 docker-7 dockerd\[4133\]:
time="2017-05-31T12:52:17.485119478Z" level=info msg="Docker daemon"
commit=89658be graphdriver=aufs

May 31 12:52:17 docker-7 systemd\[1\]: Started Docker Application
Container Engine.

May 31 12:52:17 docker-7 dockerd\[4133\]:
time="2017-05-31T12:52:17.503832254Z" level=info msg="API listen on /var/
run/docker.sock"

May 31 12:52:17 docker-7 dockerd\[4133\]:
time="2017-05-31T12:52:17.504061522Z" level=info msg="API listen on \[::
\]:4243"

$
```

```
(external)$ **curl 104.199.209.157:4243/version**

{"Version":"17.05.0-
ce","ApiVersion":"1.29","MinAPIVersion":"1.12","GitCommit":"89658be","GoVe
rsion":"go1.7.5","Os":"linux","Arch":"amd64","KernelVersion":"4.8.0-52-
generic","BuildTime":"2017-05-04T22:15:36.071254972+00:00"}

(internal)$
```

ステップ3：NetScaler CPXイメージをインストールする

Docker App StoreからNetScaler CPXイメージを取得します。CPX ExpressとCPXは同じイメージを持っています。しかし、NetScaler Consoleを使用してCPXイメージをライセンス供与してインストールすると、そのイメージは1 Gbpsのパフォーマンスを持つ完全なCPXインスタンスになります。ライセンスがない場合、イメージは20 Mbpsと250のSSL接続をサポートするCPX Expressインスタンスになります。

```
$ **sudo docker pull store/citrix/citrixadccpx:13.0-36.29**

13.0-36.29: Pulling from store/citrix/citrixadccpx

4e1f679e8ab4: Pull complete

a3ed95caeb02: Pull complete

2931a926d44b: Pull complete

362cd40c5745: Pull complete

d10118725a7a: Pull complete

1e570419a7e5: Pull complete

d19e06114233: Pull complete

d3230f008ffd: Pull complete

22bdb10a70ec: Pull complete

1a5183d7324d: Pull complete
```

```

241868d4ebff: Pull complete

3f963e7ae2fc: Pull complete

fd254cf1ea7c: Pull complete

33689c749176: Pull complete

59c27bad28f5: Pull complete

588f5003e10f: Pull complete

Digest:
sha256:31a65cfa38833c747721c6fbc142faec6051e5f7b567d8b212d912b69b4f1ebe

Status: Downloaded newer image for store/citrix/citrixadccpx:13.0-36.29

$

$ **sudo docker images**

REPOSITORY TAG IMAGE ID CREATED SIZE

store/citrix/citrixadccpx:13.0-36.29 6fa57c38803f 3 weeks ago 415MB

$

```

ステップ4：NetScaler CPXインスタンスを作成する

DockerホストにNetScaler CPXイメージをインストールします。次の例に示すように、特定のサービス用にポートを開き、NetScaler ConsoleのIPアドレスを指定します。

```

bash-2.05b\# **CHOST=${1:-localhost}**

bash-2.05b\# **echo | openssl s\_client -connect $CHOST:443 | openssl x509
-fingerprint -noout | cut -d'=' -f2**

depth=0 C = US, ST = California, L = San Jose, O = NetScaler, OU =
Internal, CN = Test Only Cert

```

```
verify error:num=18:self signed certificate
```

```
verify return:1
```

```
depth=0 C = US, ST = California, L = San Jose, O = NetScaler, OU = Internal, CN = Test Only Cert
```

```
verify return:1
```

```
DONE
```

```
24:AA:8B:91:7B:72:5E:6E:C1:FD:86:FA:09:B6:42:49:FC:1E:86:A4
```

```
bash-2.05b\#
```

```
$ **sudo docker run -dt -p 50000:88 -p 5080:80 -p 5022:22 -p 5443:443 -p 5163:161/udp -e NS\_HTTP\_PORT=5080 -e NS\_HTTPS\_PORT=5443 -e NS\_SSH\_PORT=5022 -e NS\_SNMP\_PORT=5163 -e EULA=yes -e LS\_IP=xx.xx.xx.xx -e PLATFORM=CP1000 --privileged=true --ulimit core=-1 -e NS\_MGMT\_SERVER=xx.xx.xx.xx:xxxx -e NS\_MGMT\_FINGER\_PRINT=24:AA:8B:91:7B:72:5E:6E:C1:FD:86:FA:09:B6:42:49:FC:1E:86:A4 --env NS\_ROUTABLE=false --env HOST=104.199.209.157 store/citrix/citrixadccpx:13.0-36.29**
```

```
44ca1c6c0907e17a10ffcb9ffe33cd3e9f71898d8812f816e714821870fa3538
```

```
$
```

```
$ **sudo docker ps**
```

```
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
```

```
44ca1c6c0907 store/citrix/citrixadccpx:13.0-36.29 "/bin/sh -c 'bash ..." 19 seconds ago Up 17 seconds 0.0.0.0:5022->22/tcp, 0.0.0.0:5080->80/tcp, 0.0.0.0:50000->88/tcp, 0.0.0.0:5163->161/udp, 0.0.0.0:5443->443/tcp gifted\_perlman
```

```
$
```

```
$ **ssh -p 5022 root@localhost**

root@localhost's password:

Welcome to nsoslx 1.0 (GNU/Linux 4.8.0-52-generic x86\_64)

\* Documentation: https://www.citrix.com/

Last login: Mon Jun 5 18:58:51 2017 from xx.xx.xx.xx

root@44ca1c6c0907:~\#

root@44ca1c6c0907:~\#

root@44ca1c6c0907:~\# **cli\_script.sh 'show ns ip'**

exec: show ns ip

IpAddress Traffic Domain Type Mode Arp Icmp Vserver State

\-----

1\) 172.17.0.2 0 NetScaler IP Active Enabled Enabled NA Enabled

2\) 192.0.0.1 0 SNIP Active Enabled Enabled NA Enabled

Done

root@44ca1c6c0907:~\# **cli\_script.sh 'show licenseserver'**

exec: show licenseserver

1\) ServerName: xx.xx.xx.xxPort: 27000 Status: 1 Grace: 0 Gptimeleft: 0

Done

root@44ca1c6c0907:~\# cli\_script.sh 'show capacity'

exec: show capacity
```

```
Actualbandwidth: 1000 Platform: CP1000 Unit: Mbps Maxbandwidth: 3000  
Minbandwidth: 20 Instancecount: 0
```

```
Done
```

```
root@44ca1c6c0907:~\#
```

```
$ **sudo iptables -t nat -L -n**
```

```
Chain PREROUTING (policy ACCEPT)
```

```
target prot opt source destination
```

```
DOCKER all -- 0.0.0.0/0 0.0.0.0/0 ADDRTYPE match dst-type LOCAL
```

```
Chain INPUT (policy ACCEPT)
```

```
target prot opt source destination
```

```
Chain OUTPUT (policy ACCEPT)
```

```
target prot opt source destination
```

```
DOCKER all -- 0.0.0.0/0 \!127.0.0.0/8 ADDRTYPE match dst-type LOCAL
```

```
Chain POSTROUTING (policy ACCEPT)
```

```
target prot opt source destination
```

```
MASQUERADE all -- 172.17.0.0/16 0.0.0.0/0
```

```
MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:443
```

```
MASQUERADE udp -- 172.17.0.2 172.17.0.2 udp dpt:161
```

```

MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:88

MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:80

MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:22


Chain DOCKER (2 references)

target prot opt source destination

RETURN all -- 0.0.0.0/0 0.0.0.0/0


DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5443 to:172.17.0.2:443

DNAT udp -- 0.0.0.0/0 0.0.0.0/0 udp dpt:5163 to:172.17.0.2:161

DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:50000 to:172.17.0.2:88

DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5080 to:172.17.0.2:80

DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5022 to:172.17.0.2:22

$

```

ステップ5：NetScaler Consoleを介してNetScaler CPXのライセンスを付与する

NetScaler Consoleがオンプレミスで実行されていると仮定すると、NetScaler CPXがNetScaler Consoleと通信し、情報を送信していることを検証できるはずです。次の画像は、NetScaler CPX がNetScaler Consoleからライセンスを取得している様子を示しています。



ステップ 6: NetScaler CPX で負荷分散サービスを設定し、構成を確認する

まず、Docker ホストに NGINX Web サーバーをインストールします。次に、NetScaler CPX で負荷分散を設定して2つのWebサーバーの負荷を分散させ、構成をテストします。

NGINX Web サーバーをインストールする

次の例に示すコマンドを使用して、NGINX Web サーバーをインストールします。

```
$ sudo docker pull nginx

Using default tag: latest

latest: Pulling from library/nginx

Digest:
sha256:41ad9967ea448d7c2b203c699b429abe1ed5af331cd92533900c6d77490e0268

Status: Image is up to date for nginx:latest


**$ sudo docker run -d -p 81:80 nginx**

098a77974818f451c052ecd172080a7d45e446239479d9213cd4ea6a3678616f


**$ sudo docker run -d -p 82:80 nginx**

bbdac2920bb4085f70b588292697813e5975389dd546c0512daf45079798db65


**$ sudo iptables -t nat -L -n**

Chain PREROUTING (policy ACCEPT)

target prot opt source destination

DOCKER all -- 0.0.0.0/0 0.0.0.0/0 ADDRTYPE match dst-type LOCAL
```

```
Chain INPUT (policy ACCEPT)
```

```
target prot opt source destination
```

```
Chain OUTPUT (policy ACCEPT)
```

```
target prot opt source destination
```

```
DOCKER all -- 0.0.0.0/0 \!127.0.0.0/8 ADDRTYPE match dst-type LOCAL
```

```
Chain POSTROUTING (policy ACCEPT)
```

```
target prot opt source destination
```

```
MASQUERADE all -- 172.17.0.0/16 0.0.0.0/0
```

```
MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:443
```

```
MASQUERADE udp -- 172.17.0.2 172.17.0.2 udp dpt:161
```

```
MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:88
```

```
MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:80
```

```
MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:22
```

```
MASQUERADE tcp -- 172.17.0.3 172.17.0.3 tcp dpt:80
```

```
MASQUERADE tcp -- 172.17.0.4 172.17.0.4 tcp dpt:80
```

```
Chain DOCKER (2 references)
```

```
target prot opt source destination
```

```
RETURN all -- 0.0.0.0/0 0.0.0.0/0
```

```
DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5443 to:172.17.0.2:443
```

```
DNAT udp -- 0.0.0.0/0 0.0.0.0/0 udp dpt:5163 to:172.17.0.2:161

DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:50000 to:172.17.0.2:88

DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5080 to:172.17.0.2:80

DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5022 to:172.17.0.2:22

DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:81 to:172.17.0.3:80

DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:82 to:172.17.0.4:80

$
```

NetScaler CPX で負荷分散を設定し、両方のWebサービスへの負荷分散を確認する

```
$ **ssh -p 5022 root@localhost**

root@localhost's password:

Welcome to nsoslx 1.0 (GNU/Linux 4.8.0-52-generic x86\_64)

\* Documentation: https://www.citrix.com/

Last login: Mon Jun 5 18:58:54 2017 from 172.17.0.1

root@44ca1c6c0907:~\#

root@44ca1c6c0907:~\#

root@44ca1c6c0907:~\#

root@44ca1c6c0907:~\#

root@44ca1c6c0907:~\#

root@44ca1c6c0907:~\# **cli\_script.sh "add service web1 172.17.0.3 HTTP
80"**

exec: add service web1 172.17.0.3 HTTP 80

Done
```

```
root@44ca1c6c0907:~\# **cli\_script.sh "add service web2 172.17.0.4 HTTP 80" **
```

```
exec: add service web2 172.17.0.4 HTTP 80
```

Done

```
root@44ca1c6c0907:~\# **cli\_script.sh "add lb vserver cpx-vip HTTP 172.17.0.2 88" **
```

```
exec: add lb vserver cpx-vip HTTP 172.17.0.2 88
```

Done

```
root@44ca1c6c0907:~\# **cli\_script.sh "bind lb vserver cpx-vip web1" **
```

```
exec: bind lb vserver cpx-vip web1
```

Done

```
root@44ca1c6c0907:~\# **cli\_script.sh "bind lb vserver cpx-vip web2" **
```

```
exec: bind lb vserver cpx-vip web2
```

Done

```
root@44ca1c6c0907:~\#
```

```
root@44ca1c6c0907:~\# **cli\_script.sh 'show lb vserver cpx-vip' **
```

```
exec: show lb vserver cpx-vip
```

cpx-vip (172.17.0.2:88) - HTTP Type: ADDRESS

State: UP

Last state change was at Mon Jun 5 19:01:49 2017

Time since last state change: 0 days, 00:00:42.620

Effective State: UP

Client Idle Timeout: 180 sec

Down state flush: ENABLED

Disable Primary Vserver On Down : DISABLED

Appflow logging: ENABLED

Port Rewrite : DISABLED

No. of Bound Services : 2 (Total) 2 (Active)

Configured Method: LEASTCONNECTION

Current Method: Round Robin, Reason: A new service is bound BackupMethod: ROUNDROBIN

Mode: IP

Persistence: NONE

Vserver IP and Port insertion: OFF

Push: DISABLED Push VServer:

Push Multi Clients: NO

Push Label Rule: none

L2Conn: OFF

Skip Persistency: None

Listen Policy: NONE

IcmpResponse: PASSIVE

RHlstate: PASSIVE

New Service Startup Request Rate: 0 PER_SECOND, Increment Interval: 0

Mac mode Retain Vlan: DISABLED

DBS_LB: DISABLED

Process Local: DISABLED

Traffic Domain: 0

TROFS Persistence honored: ENABLED

Retain Connections on Cluster: NO

2\) web1 (172.17.0.3: 80) - HTTP State: UP Weight: 1

3\) web2 (172.17.0.4: 80) - HTTP State: UP Weight: 1

Done

root@44ca1c6c0907:~\#

(external)\$ **curl 104.199.209.157:50000**

\<\!DOCTYPE html\>

\<html\>

\<head\>

\<title\>Welcome to nginx\!\</title\>

\<style\>

body {

width: 35em;

margin: 0 auto;

font-family: Tahoma, Verdana, Arial, sans-serif;

```
}

\</style\>

\</head\>

\<body\>

\<h1\>Welcome to nginx\!\</h1\>

\<p\>If you see this page, the nginx web server is successfully installed
and

working. Further configuration is required.\</p\>


\<p\>For online documentation and support please refer to

\<a href="http://nginx.org/"\>nginx.org\</a\>.\<br/\>

Commercial support is available at

\<a href="http://nginx.com/"\>nginx.com\</a\>.\</p\>


\<p\>\<em\>Thank you for using nginx.\</em\>\</p\>

\</body\>

\</html\>

(external)$

(external)$ for i in {1..100} ; **do curl http://104.199.209.157:50000
-o /dev/null ; done**
```

% Total		% Received		% Xferd		Average Speed		Time	Time	Time
Current										
						Dload	Upload	Total	Spent	Left
Speed										
100	612	100	612	0	0	1767	0	--:--:--	--:--:--	--:--:--
1768										

% Total		% Received		% Xferd		Average Speed		Time	Time	Time
Current										
						Dload	Upload	Total	Spent	Left
Speed										
100	612	100	612	0	0	1893	0	--:--:--	--:--:--	--:--:--
1894										

% Total		% Received		% Xferd		Average Speed		Time	Time	Time
Current										
						Dload	Upload	Total	Spent	Left
Speed										
100	612	100	612	0	0	1884	0	--:--:--	--:--:--	--:--:--
1883										

% Total		% Received		% Xferd		Average Speed		Time	Time	Time
Current										
						Dload	Upload	Total	Spent	Left
Speed										
100	612	100	612	0	0	1917	0	--:--:--	--:--:--	--:--:--
1924										

% Total		% Received		% Xferd		Average Speed		Time	Time	Time
Current										
						Dload	Upload	Total	Spent	Left
Speed										
100	612	100	612	0	0	1877	0	--:--:--	--:--:--	--:--:--
1883										

% Total		% Received		% Xferd		Average Speed		Time	Time	Time
Current										
						Dload	Upload	Total	Spent	Left
Speed										
100	612	100	612	0	0	1852	0	--:--:--	--:--:--	--:--:--
1848										

% Total		% Received		% Xferd		Average Speed		Time	Time	Time
Current										
						Dload	Upload	Total	Spent	Left
Speed										
100	612	100	612	0	0	1860	0	--:--:--	--:--:--	--:--:--
1865										

% Total		% Received		% Xferd		Average Speed		Time	Time	Time
Current										
						Dload	Upload	Total	Spent	Left
Speed										
100	612	100	612	0	0	1887	0	--:--:--	--:--:--	--:--:--
1888										

% Total		% Received		% Xferd		Average Speed		Time	Time	Time
Current										
						Dload	Upload	Total	Spent	Left
Speed										
100	612	100	612	0	0	1802	0	--:--:--	--:--:--	--:--:--

1800

% Total	% Received	% Xferd	Average	Speed	Time	Time	Time
Current							

			Dload	Upload	Total	Spent	Left
Speed							

100	612	100	612	0	0	1902	0	--:--:--	--:--:--	--:--:--
1906										

% Total	% Received	% Xferd	Average	Speed	Time	Time	Time
Current							

			Dload	Upload	Total	Spent	Left
Speed							

100	612	100	612	0	0	1843	0	--:--:--	--:--:--	--:--:--
1848										

% Total	% Received	% Xferd	Average	Speed	Time	Time	Time
Current							

			Dload	Upload	Total	Spent	Left
--	--	--	-------	--------	-------	-------	------

Speed

100	612	100	612	0	0	1862	0	--:--:--	--:--:--	--:--:--
1860										

% Total	% Received	% Xferd	Average	Speed	Time	Time	Time
Current							

Dload	Upload	Total	Spent	Left
-------	--------	-------	-------	------

Speed

100	612	100	612	0	0	1806	0	--:--:--	--:--:--	--:--:--
1810										

% Total	% Received	% Xferd	Average	Speed	Time	Time	Time
Current							

Dload	Upload	Total	Spent	Left
-------	--------	-------	-------	------

Speed

100	612	100	612	0	0	1702	0	--:--:--	--:--:--	--:--:--
1704										

(external)\$

```
root@44ca1c6c0907:~\# **cli\_script.sh 'stat lb vserver cpx-vip'**
```

```
exec: stat lb vserver cpx-vip
```

Virtual Server Summary

		vsvrIP	port	Protocol	State	Health
actSvcs						
cpx-vip		172.17.0.2	88	HTTP	UP	
100	2					

inactSvcs

cpx-vip	0
---------	---

Virtual Server Statistics

	Rate (/s)	Total
Vserver hits	0	101
Requests	0	101
Responses	0	101
Request bytes	0	8585
Response bytes	0	85850
Total Packets rcvd	0	708
Total Packets sent	0	408
Current client connections	--	0

Current Client Est connections	--	0
Current server connections	--	0
Current Persistence Sessions	--	0
Requests in surge queue	--	0
Requests in vserver's surgeQ	--	0
Requests in service's surgeQs	--	0
Spill Over Threshold	--	0
Spill Over Hits	--	0
Labeled Connection	--	0
Push Labeled Connection	--	0
Deferred Request	0	0

Invalid Request/Response	--	0
Invalid Request/Response Dropped	--	0
Vserver Down Backup Hits	--	0
Current Multipath TCP sessions	--	0
Current Multipath TCP subflows	--	0
Apdex for client response times.	--	1.00
Average client TTLB	--	0
web1 0/s	172.17.0.3 80 HTTP UP	51
web2 0/s	172.17.0.4 80 HTTP UP	50
Done		

```
root@44ca1c6c0907:~\#
```

NetScaler CPX のトラブルシューティング

このドキュメントでは、NetScaler CPX の使用中に発生する可能性のある問題のトラブルシューティング方法について説明します。このドキュメントを使用すると、ログを収集して原因を特定し、NetScaler CPX のインストールと構成に関連する一般的な問題のいくつかの回避策を適用できます。

- NetScaler CPX のログを表示するにはどうすればよいですか？

NetScaler CPX が `tty:true` オプションで展開されている場合、`kubectl logs` コマンドを使用して NetScaler CPX のログを表示できます。ログを表示するには、次のコマンドを実行します。

```
kubectl logs <pod-name> [-c <container-name>] [-n <namespace-name>]
```

例：

```
kubectl logs cpx-ingress1-69b9b8c648-t8bgn -c cpx -n citrix-adc
```

`tty:true` オプションを使用した NetScaler CPX ポッド展開の例を次に示します。

```
containers:
  - name: cpx-ingress
    image: "quay.io/citrix/citrix-k8s-cpx-ingress:13.0-58.30"
    tty: true
    securityContext:
      privileged: true
    env:
```

さらに多くのブートログは、NetScaler CPX ファイルシステムの `/cpx/log/boot.log` ファイルにあります。

注：ポッド名を取得するには、`kubectl get pods -o wide` コマンドを実行します。

- NetScaler CPX からテクニカルサポートバンドルを収集するにはどうすればよいですか？

NetScaler CPX テクニカルサポートバンドルを収集するには、Kubernetes マスターノードのシェルインターフェイスで次のコマンドを実行します。

```
kubectl exec <cpx-pod-name> [-c <cpx-container-name>] [-n
<namespace-name>] /var/netscaler/bins/cli_script.sh "show
techsupport"
```

テクニカルサポートバンドルは、NetScaler CPX のファイルシステムの `/var/tmp/support` ディレクトリに表示されます。NetScaler CPX から目的の宛先にテクニカルサポートバンドルをコピーするには、`scp` または `kubectl cp` を使用します。

例：

```
root@localhost# kubectl exec cpx-ingress1-55b9b6fc75-t5kc6 -c cpx -n
citrix-adc /var/netscaler/bins/cli_script.sh "show techsupport"

exec: show techsupport

Scope:  NODE

Done

root@localhost# kubectl cp cpx-ingress1-55b9b6fc75-t5kc6:var/tmp/
support/collector_P_192.168.29.232_31Aug2020_07_30.tar.gz /tmp/
collector_P_192.168.29.232_31Aug2020_07_30.tar.gz -c cpx

root@localhost# ll /tmp/
collector_P_192.168.29.232_31Aug2020_07_30.tar.gz

-rw-r--r-- 1 root root 1648109 Aug 31 13:23 /tmp/
collector_P_192.168.29.232_31Aug2020_07_30.tar.gz
```

- NetScaler CPX ポッドが起動中にスタックするのはなぜですか？

`kubectl describe pods` コマンドを使用してポッドのステータスを確認できます。ポッドのステータスを知るには、次のコマンドを実行します。

```
kubectl describe pods <pod-name> [-c <container-name>] [-n
<namespace-name>]
```

例：

```
kubectl describe pods cpx-ingress1-69b9b8c648-t8bgn
```

ポッドイベントでコンテナが起動していると表示された場合は、ポッドログを確認する必要があります。

- NetScaler CPXポッドとKubernetesマスターノード間でファイルをコピーするにはどうすればよいですか？

ホストのファイルシステムに `/cpx` ディレクトリをマウントするには、Dockerのボリュームマウント機能を使用することをお勧めします。NetScaler CPXコンテナがコアダンプで終了した場合、ログやその他の重要なデータはマウントポイントで利用できます。

NetScaler CPXポッドとKubernetesマスターノード間でファイルをコピーするには、以下のいずれかのコマンドを使用できます。

kubectl cp: ポッドからノードにファイルをコピーするには、次のコマンドを実行します。

```
kubectl cp <pod-name>:<absolute-src-path> <dst-path> [-c
<container-name>] [-n <namespace-name>]
```

例:

```
root@localhost:~# kubectl cp cpx-ingress-596d56bb6-zbx6h:cpx/log/
boot.log /tmp/cpx-boot.log -c cpx-ingress

root@localhost:~# ll /tmp/cpx-boot.log

-rw-r--r-- 1 root root 7880 Sep 11 00:07 /tmp/cpx-boot.log
```

scp: このコマンドを使用して、NetScaler CPXポッドとKubernetesノード間でファイルをコピーできます。ポッドからノードにファイルをコピーするには、次のコマンドを実行します。パスワードの入力を求められたら、SSHユーザーのパスワードを入力します。

```
scp <user>@<pod-ip>:<absolute-src-path> <dst-path>
```

例:

```
root@localhost:~# scp nsroot@192.168.29.198:/cpx/log/boot.log /tmp/cpx-
boot.log

nsroot@192.168.29.198's password:

boot.log

100% 7880      5.1MB/s   00:00

root@localhost:~#
```

- NetScaler CPXでパケットをキャプチャするにはどうすればよいですか？

NetScaler CPXでパケットをキャプチャするには、`kubectl exec` コマンドを使用して NetScaler CPXのシェルインターフェースを起動します。NetScaler CPXポッドのシェルインターフェースを起動するには、次のコマンドを実行します。

```
kubectl exec -it pod-name [-c container-name] [-n namespace-name] bash
```

例:

```
kubectl exec -it cpx-ingress1-69b9b8c648-t8bgn -c cpx -n citrix-adc
bash
```

そして、パケットキャプチャを開始するには、次のコマンドを実行します。

```
cli_script.sh "start nstrace -size 0"
```

進行中のパケットキャプチャを停止したい場合は、次のコマンドを実行します。

```
cli_script.sh "stop nstrace"
```

キャプチャされたパケットは、NetScaler CPXファイルシステムの*/cpx/nstrace/time-stampディレクトリにある.cap*ファイルで表示できます。

- NetScaler CPXが `LS_IP=<ADM-IP>` 環境変数でデプロイされているにもかかわらず、ライセンスサーバーが構成されないのはなぜですか？

NetScaler CPXが展開されているノードからライセンスサーバーにアクセスできることを確認してください。 `ping <ADM-IP>` コマンドを使用して、NetScaler CPXノードからNetScaler Consoleへの接続を確認できます。

NetScaler Consoleがノードからアクセスできる場合、`/cpx/log/boot.log`ファイルでライセンスサーバー構成ログを確認する必要があります。NetScaler CPXポッドのシェルインターフェースで次のコマンドを使用して、ライセンスサーバー構成を確認することもできます。

```
cli_script.sh "show licenseserver"
```

例:

```
root@cpx-ingress-596d56bb6-zbx6h:/cpx/log# cli_script.sh "show
licenseserver"

exec: show licenseserver

ServerName:  10.106.102.199Port:  27000 Status:  1   Grace:  0
  Gptimeleft:  720

Done
```

- NetScaler CPXでライセンスサーバーの構成が成功した後でも、プールされたライセンスがNetScaler CPXで構成されないのはなぜですか？

`/cpx/log/boot.log`ファイルでライセンス構成ログを確認してください。NetScaler CPXポッドのシェルインターフェースで次のコマンドを使用して、NetScaler CPXで構成されたプールされたライセンスを確認することもできます。

```
cli_script.sh "show capacity"
```

例,

```
root@cpx-ingress-596d56bb6-zbx6h:/cpx/log# cli_script.sh "show
capacity"

exec: show capacity
```

```
Actualbandwidth: 1000 MaxVcpuCount: 2 Edition: Platinum Unit:
Mbps Bandwidth: 0 ` `Maxbandwidth: 40000 Minbandwidth: 20
Instancecount: 1
```

```
Done
```

また、必要なライセンスファイルがライセンスサーバーにアップロードされていることを確認してください。NetScaler CPXでライセンスサーバーが正常に構成された後、次のコマンドを使用してライセンスサーバーで利用可能なライセンスを確認することもできます。NetScaler CPXポッドのシェルインターフェースでコマンドを実行します。

```
cli_script.sh "sh licenseserverpool"
```

例:

```
root@cpx-ingress-596d56bb6-zbx6h:/cpx/log# cli_script.sh "show
licenseserverpool"
```

```
exec: show licenseserverpool
```

```
Instance Total                : 5

Instance Available            : 4

Standard Bandwidth Total      : 0 Mbps

Standard Bandwidth Availabe    : 0 Mbps

Enterprise Bandwidth Total     : 0 Mbps

Enterprise Bandwidth Available : 0 Mbps

Platinum Bandwidth Total      : 10.00 Gbps

Platinum Bandwidth Available   : 9.99 Gbps

CP1000 Instance Total         : 100

CP1000 Instance Available     : 100
```

Done

- NITRO API呼び出しがNetScaler CPXから*Connection Refused*応答を受け取るのはなぜですか？

NetScaler CPXリリース12.1以降、NITRO APIのデフォルトポートは9080（非セキュア）および9443（セキュア）です。アクセスしようとしているNetScaler CPXのNITROポートがポッドで公開されていることを確認してください。 `kubectl describe` コマンドを実行して、NetScaler CPXコンテナセクションでNetScaler CPXコンテナの公開およびマッピングされたポートを表示できます。

```
kubectl describe pods <pod-name> | grep -i port
```

例:

```
ng472 | grep -i port
```

```
Ports:          80/TCP, 443/TCP, 9080/TCP, 9443/TCP
```

```
Host Ports:     0/TCP, 0/TCP, 0/TCP, 0/TCP
```

```
NS_HTTP_PORT:   9080
```

```
NS_HTTPS_PORT:  9443
```

```
Port:          <none>
```

```
Host Port:     <none>
```

```
NS_PORT:       80
```

- トラフィックがない、またはほとんどない場合でも、NetScaler CPXのNSPPEプロセスがCPU使用率のほとんどを消費するのはなぜですか？

NetScaler CPXが `CPX_CONFIG='{ "YIELD": "NO" }'` 環境変数で展開されている場合、トラフィックがない、またはほとんどない場合でも、NSPPEプロセスはCPU使用率の100パーセントを消費します。NSPPEプロセスがCPU使用率を消費しないようにしたい場合は、

`CPX_CONFIG='{ "YIELD": "NO" }'` 環境変数なしでNetScaler CPXを展開する必要があります。

デフォルトでは、CPXのNSPPEプロセスはCPU使用率を独占したり消費したりしないように構成されています。

- NetScaler Consoleに登録するために必要な環境変数で展開されたにもかかわらず、NetScaler CPXがNetScaler Consoleにリストされないのはなぜですか？

NetScaler CPXのNetScaler Consoleへの登録ログは、NetScaler CPXファイルシステムの*/cpx/log/boot.log*ファイルで見つけることができます。

NetScaler CPXポッドから `ping` コマンドを使用して、NetScaler Console IPアドレスのアクセシビリティを確認できます。また、NetScaler Console登録に必要なすべての環境変数がNetScaler CPXコンテナに設定されていることを確認してください。

- `NS_MGMT_SERVER` : ADM-IPアドレスまたはFQDNを指定します。
 - `HOST` : ノードIPアドレスを指定します。
 - `NS_HTTP_PORT` : ノード上のマッピングされたHTTPポートを指定します。
 - `NS_HTTPS_PORT` : ノード上のマッピングされたHTTPSポートを指定します。
 - `NS_SSH_PORT` : ノード上のマッピングされたSSHポートを指定します。
 - `NS_SNMP_PORT` : ノード上のマッピングされたSNMPポートを指定します。
 - `NS_ROUTABLE` : NetScaler CPXポッドのIPアドレスは外部からルーティングできません。
 - `NS_MGMT_USER` : ADMユーザー名を指定します。
 - `NS_MGMT_PASS` : ADMパスワードを指定します。
- nsrootユーザーのパスワードを変更した後、`cli_script.sh` に「*Invalid user name or password*」というエラーメッセージが表示されるのはなぜですか？

`cli_script.sh` コマンドは、NetScaler CPX上のNSCLIのラッパーユーティリティです。最初の引数をコマンド文字列またはファイルパスとして実行し、2番目の引数はオプションで資格情報です。nsrootユーザーのパスワードが変更された場合、`cli_script.sh` の2番目の引数として資格情報を提供する必要があります。資格情報を使用してNSCLIを実行するには、次のコマンドを実行します。

```
cli_script.sh "<command>" ":<username>:<password>"
```

例:

```

root@087a1e34642d:/# cli_script.sh "show ns ip"

exec: show ns ip

ERROR: Invalid username or password

root@087a1e34642d:/# cli_script.sh "show ns ip" ":nsroot:nsroot123"

exec: show ns ip


```

IpAddress	Traffic Domain	Type	Mode	Arp
Icmp	Vserver	State		
-----	----	----	---	----
-----	-----	-----		
172.17.0.3	0	NetScaler IP	Active	Enabled
Enabled NA	Enabled			
192.0.0.1	0	SNIP	Active	Enabled
Enabled NA	Enabled			
Done				

- `root` および `nsroot` ユーザーでNetScaler CPXへのSSHが失敗するのはなぜですか？

13.0-64.35リリース以降、NetScaler CPXはデフォルトのパスワードを生成し、SSHユーザー（`root` および `nsroot`）用に更新します。パスワードを手動で変更していない場合、SSHユーザーのパスワードはNetScaler CPXのファイルシステム上の `/var/deviceinfo/random_id` で見つけることができます。

fix

This is the content for the sample heading topic. It is hidden from the main table of contents but remains accessible via links.