

NetScaler CPX 14.1

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での単一IPアドレスの仕組み	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 エクスプレス	12
商用NetScaler CPX	13
NetScaler CPXインスタンスでサポートされるプール容量	13
NetScaler CPXライセンスはどのように機能しますか？	13
NetScaler CPXのライセンス供与方法？	14
LASベースのCPXライセンスの開始	14
NetScaler CPXライセンスの問題のトラブルシューティング	16
NetScaler CPX の Kubernetes プローブのサポート	17
NetScaler CPX の Kubernetes プローブを構成する	17
CLI を使用してプローブを構成する	18
DockerでのNetScaler CPXインスタンスの展開	21
前提条件	21
QuayからNetScaler CPXイメージをダウンロードする	22
docker runコマンドを使用したNetScaler CPXインスタンスの展開	22
docker runコマンドを使用したNetScaler CPXの軽量バージョンの展開	27
Docker Compose を使用した NetScaler CPX インスタンスの展開	28
NetScaler CPX インスタンスを NetScaler Console に追加する	31

前提条件	31
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	32
Kubernetes ConfigMapを使用してNetScaler ConsoleにNetScaler CPXインスタンスを追加する	33
NetScaler CPX ライセンスアグリゲーター	37
NetScaler CPXライセンスアグリゲーターの主な利点	37
展開トポロジ	38
Helm チャートを使用した NetScaler CPX License Aggregator の展開	38
前提条件	38
Helm チャートを使用した展開	39
Helm チャートのインストール	39
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	39
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	40
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	41
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	42
NetScaler CPX の構成	45
コマンドラインインターフェイスを使用して NetScaler CPX インスタンスを構成する	45
NetScaler CPX で非デフォルトパスワードを使用するためのサポート	45
NITRO API を使用した NetScaler CPX インスタンスの構成	46
ジョブを使用した NetScaler CPX インスタンスの構成	46
NetScaler CPXインスタンスでのAppFlowの構成	48
AppFlow®を有効にする	48
コレクターを指定する	49
AppFlowアクションの構成	50
AppFlowポリシーの構成	50
AppFlowポリシーのバインド	51
構成ファイルを使用してNetScaler CPXを構成する	52

静的構成ファイルの構造	52
NetScaler CPXにおける動的ルーティングのサポート	55
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	55
Dockerロギングドライバーの構成	59
NetScaler CPXインスタンスのアップグレード	60
前提条件	60
NetScaler CPXインスタンスをアップグレードするには	61
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	63
East-Westトラフィックフローを有効にするプロキシとしてNetScaler CPXを展開	65
シングルホストネットワークにNetScaler CPXを展開する	68
マルチホストネットワークでのNetScaler CPXの展開	70
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	70
トポロジ 2: 物理サーバーとクライアントを備えた NetScaler CPX	72
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	73
ネットワークに直接アクセスしてNetScaler CPXを展開する	76
ConfigMapを使用してKubernetesでNetScaler CPXを構成する	78
ConfigMap を使用した NetScaler CPX のカスタムメトリクス設定	84
カスタムメトリクスを構成する	84
CLI を使用して構成するには	84
YAML ファイルを使用して構成するには	86
KubernetesノードのローカルDNSキャッシュとしてNetScaler CPXを展開する	88
ノードにNetScaler CPXをDNSキャッシュとして展開する	89
Kubernetesマスターノードでの設定	89
Kubernetesクラスター内のワーカーノードでの設定	94
Google Compute Engine に NetScaler CPX プロキシを展開する	96
前提条件	96
構成手順	96

ステップ1：GCE VMにDockerをインストールする	97
ステップ2：DockerインスタンスとのリモートAPI通信を設定する	108
ステップ3：NetScaler CPXイメージをインストールする	111
ステップ4：NetScaler CPXインスタンスを作成する	112
ステップ5：NetScaler Consoleを介してNetScaler CPXのライセンスを付与する	116
ステップ 6: NetScaler CPX で負荷分散サービスを構成し、構成を確認する	116
NGINX Web サーバーをインストールする	117
負荷分散のために NetScaler CPX を構成し、両方の Web サービスへの負荷の分散を確認する	119
NetScaler CPX のトラブルシューティング	134
fix	144

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向けに 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。Docker ホスト上に、1 つまたは複数の NetScaler CPX インスタンスを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トラフィックステアリング
 - サー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書き換えおよびレスポンス
 - L4 DoS防御
 - L7 DoS防御
 - Webアプリケーションファイアウォール (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。
- 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
 - ウェブログ
 - AppFlow®
 - NetScaler® アプリケーション配信管理
 - 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ルーティングおよびルートヘルスインジェクション (RHI)

注:

Rx、Tx、GRO、GSO、LROなどのインターフェイス機能は、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、こ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 OpenShift
- 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)
 - Azure Kubernetes サービス (AKS)
 - Google Kubernetes エンジン (GKE)
- ランチャー
- Pivotal Container 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*) に直接接続されます。Dockerエンジンはまた、NetScaler CPXインスタンスに172.17.0.0/16ネットワーク内の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での単一IPアドレスの仕組み

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のサブネット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単一の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、異なるポート番号を使用してNSIP、SNIP、およびVIPとして機能します。

次の図は、単一の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。



この例では、pingリクエストは、送信元IPアドレスをNetScaler CPX IPアドレス (172.17.0.4) として、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ー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なかった場合は、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IPアドレスとポートにトラフィックを誘導する仕組みを示しています。



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NetScaler CPXが必要です。NetScaler CPXは、従来のプールライセンスと新しい[Citrix Platform or Universal Hybrid Multi-Cloud licensing](#)をサポートしています。すべてのNetScaler CPXインスタンスは、NetScaler Console（旧NetScaler ADM）のみを介してライセンスされます。

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容量、Universal Hybrid Multi-Cloud (UHMC)、またはCitrix Platform License (CPL) ライセンスを管理できます。

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

以下の種類のNetScaler CPXライセンスがサポートされています。

- NetScaler CPX CPX Express - 無料、Kubernetes向け開発者中心NetScaler
- 商用NetScaler CPX – 本番環境対応、エンタープライズサポートプロキシ

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で無償で利用できる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、NetScaler CPX ExpressライセンスがデフォルトでNetScaler CPXインスタンスに適用されます。NetScaler NetScaler CPX Expressライセンスでは、すべてのNetScaler CPX機能が利用可能です。CPXでサポートされている機能については、[NetScaler CPXについて](#)を参照してください。NetScaler CPX ExpressではNetScalerコミュニティサポートが利用できますが、エンタープライズサポートは利用できません。

NetScaler CPX Expressライセンスは100 Mbpsの帯域幅容量を提供します。より高い容量、本番環境への展開、およびエンタープライズサポートを利用するには、NetScaler CPXインスタンスのライセンスが必要です。

注:

NetScaler CPXバージョン14.1ビルド17.x以降、NetScaler CPX Expressライセンスの帯域幅は20 Mbpsから100 Mbpsに増加しました。この帯域幅の増加により、Kubernetes開発者はNetScalerプロキシを試すことができます。

商用NetScaler CPX

LASは、April 15, 2026以降、NetScalerインスタンスをアクティブ化およびライセンス供与する唯一の方法となり、NetScaler Flexedライセンス (CPL/UHMC)、従来のNetScaler帯域幅プールライセンスをサポートします。LAS移行後、NetScaler CPXはNetScaler Flexedライセンス (CPL/UMC) または従来のNetScalerプール帯域幅ライセンスのみをサポートします。詳細については、[フレックスライセンスへの移行](#)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容量

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制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のマーケットプレイスで利用できるようになります。

NetScaler CPXライセンスはどのように機能しますか？

NetScaler CPXプール/CPL/UHMCライセンス： 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借り出すことが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返却し、必要とする他のインスタンスが利用できるようにします。

NetScaler CPXの動作： 最大1 Gbpsのスループットを借り出す単一のNetScaler CPXインスタンスは、インスタンスプールからのみ借り出し、帯域幅ライセンスプールからは借り出しません。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、NetScaler 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使用され、残りの200 Mbpsは帯域幅プールから消費されます。



NetScaler CPXライセンスのワークフロー：

1. LASライセンスのために、NetScalerエージェントサービスまたはCitrix Cloudと連携するオンプレミスのNetScalerエージェントをセットアップします。
2. Console Containerエージェントを展開し、NetScalerエージェントサービスまたはオンプレミスのNetScalerエージェントでそのステータスを確認します。
3. NetScaler CPXライセンスを設定し、NetScaler CPXを展開します。
 - a. 起動時に、NetScaler CPXはライセンス登録要求をConsoleコンテナエージェントに送信し、ConsoleコンテナエージェントはそれをNetScalerエージェントサービスまたはオンプレミスのNetScalerエージェントに中継します。
 - b. NetScalerエージェントサービスまたはオンプレミスのNetScalerエージェントは、NetScaler CPXを登録し、帯域幅を割り当てます。
 - c. プール/UHMC/CPLライセンスプロブが利用可能な場合、ライセンスが付与されます。

NetScaler CPXのライセンス供与方法？

NetScaler CPXは、NetScalerエージェントサービスまたはオンプレミスを通じてライセンスされます。Kubernetesクラスター内のNetScaler CPXインスタンスにライセンスを付与するには、単一のConsoleコンテナエージェントを展開します。クラスターごとに1つのエージェントが、そのクラスター内のすべてのNetScaler CPXインスタンスにライセンスを付与します。

LASベースのCPXライセンスの開始

1. 前提条件：[NetScaler Console Service](#)または[NetScaler Console](#)オンプレミスでLASベースのライセンスが有効になっている必要があります。

2. NetScaler CPXインスタンスが展開されているKubernetesクラスターに、コンテナベースのコンソールエージェントを展開します。
 - a. NetScaler Consoleサービスのお客様向けにコンテナベースのコンソールエージェントを展開する手順は次のとおりです。
 - i. [HELM chart](#)を使用する
 - ii. [OpenShift Operator](#)を使用する
 - b. NetScaler Consoleオンプレミスのお客様向けにコンテナベースのコンソールエージェントを展開する手順は次のとおりです。
 - i. [YAMLs](#)を使用する。
 - ii. [OpenShift Operator](#)を使用する。
3. LASベースのライセンスをサポートするには、NetScaler CPX 14.1–51.x以降にアップグレードしてください。最新のNetScaler CPX展開については、CPX HELMチャートを参照してください。LASベースのライセンスをサポートするために、CPX HELMチャートの `values.yaml` から以下の必須変数のみを更新してください。
 - a. `licensingMode` : `las`
 - b. `ADMIP`: `<Console Container agent Service IP>` - NetScaler Consoleオンプレミスまたはサービスの**Infrastructure > Instances Dashboard > Agents**セクションからコンテナエージェントのIPアドレスをコピーします。

または

`ADMIP`: `<Console Container agent ServiceName.Namespace>` - NetScaler Consoleオンプレミスまたはサービスの**Infrastructure > Instances Dashboard > Agents**セクションからコンテナエージェントの `HOSTNAME` をコピーし、コンテナエージェントが展開されている名前空間を付加します。
 - c. `loginSecret` : NetScaler Consoleコンテナエージェントのログイン資格情報詳細。
 - i. YAML/HELMベースの展開の場合、NetScaler ConsoleオンプレミスまたはサービスからコンソールエージェントのYAMLを生成する際に提供されたアプリケーションIDとパスワードの詳細に基づいて、Kubernetesシークレットを作成する必要があります。
 - ii. OpenShift Operatorベースの展開では、NetScaler Consoleログインシークレットをすでに作成している必要があります (`loginSecret` フィールドを参照)。ここで同じシークレット名を使用できます。
 - iii. `bandWidth`: Mbps単位の値。例: 500 Mbps。

NetScaler CPXライセンス(/en-us/cpx/media/cpx-licensing.png)

- d. NetScaler CPXのライセンスが正常に適用されると、NetScaler ConsoleオンプレミスまたはサービスダッシュボードにNetScaler CPXインスタンスが表示され始めます。

NetScaler CPXライセンスダッシュボード(/en-us/cpx/media/cpx-licensing-dashboard.png)

NetScaler CPXライセンスの問題のトラブルシューティング

- NetScaler CPXのライセンスを確認します。
 - `oc exec -it <CPX POD name> --bash` を使用してNetScaler CPXにログインし、`cli_script.sh "sh capacity"` を実行して実際の帯域幅の値を表示します。
 - `cli_script.sh "sh license"` を実行して `Licensing mode: LAS` の値を確認します。
- NetScaler ConsoleサービスまたはオンプレミスでライセンスされたNetScaler CPXインスタンスを見つける。
 - NetScaler ConsoleサービスまたはオンプレミスのGUIで、**NetScaler Licensing > Flexed licensing > Dashboard**に移動し、NetScaler Consoleフォームファクターを検索します。

NetScaler CPX の Kubernetes プローブのサポート

Kubernetes は、アプリケーションが正しく初期化され、正常な状態を維持するように、起動プローブ、活性プローブ、および準備プローブを通じてコンテナのライフサイクルを管理します。NetScaler CPX は、既存の起動プローブと活性プローブに加えて、準備プローブをサポートするようになりました。このサポートにより、Kubernetes は NetScaler CPX が完全に動作可能であり、設定されている場合はライセンスチェックアウトが正常に完了した場合にのみ、トラフィックを NetScaler CPX にルーティングします。

これらのプローブを使用することで、アプリケーションの可用性を向上させ、障害が発生したインスタンスや応答しないインスタンスが迅速に再作成されるようにすることができます。

利点の一部を以下に示します。

- 強化されたトラフィック管理: NetScaler CPX がリクエストを処理する準備ができたときにのみトラフィックを受信するようにします。
- 運用上の信頼性: 異常なコンテナを自動的に再起動して、障害から回復します。
- ライセンス検証: `LAS_CPX_BANDWIDTH` パラメータを使用する場合、ライセンスチェックアウトが成功するまで準備状態を遅延させます。
- 正確な初期化: コンテナの起動からアクティブなヘルス監視への移行を調整します。

注:

準備プローブを使用するには、NetScaler CPX バージョン `14.1-72.x` 以降が必要です。プローブのメカニズムに関する詳細については、公式の Kubernetes ドキュメントを参照してください。

NetScaler CPX の Kubernetes プローブを構成する

NetScaler CPX のプローブを構成するには、Kubernetes デプロイ YAML ファイルでプローブパラメータを定義します。

CLI を使用してプローブを構成する

デプロイ YAML ファイルを編集して、Kubernetes が NetScaler CPX コンテナの状態を検証するために使用する特定の `exec` コマンドを含めます。

起動プローブの構成: Kubernetes が初期化プロセスの完了を待機するように、次のスニペットを追加します。NetScaler CPX は、`{{tmp/cpx_started}}` ファイルを作成することで、起動が成功したことを通知します。活性チェックが開始される前に初期化が完了するように、このパスを監視するように起動プローブを構成します。

```
startupProbe:

  exec:

    command:

      - ls

      - /tmp/cpx_started

  initialDelaySeconds: 30

  periodSeconds: 5

  failureThreshold: 20

  successThreshold: 1

  timeoutSeconds: 1
```

レディネスプローブを構成する: ポッドがトラフィックを受け入れる準備ができたタイミングを定義するために、次のスニペットを追加します。CPXは、トラフィックを処理する準備ができたときに `/tmp/cpx_ready` ファイルを作成します。 `LAS_CPX_BANDWIDTH` パラメータを構成した場合、CPXはライセンスチェックアウトが成功した後にこのファイルを作成します。それ以外の場合は、起動プロセスが完了した後に表示されます。Kubernetesは、レディネスプローブのためにこのファイルを使用して、ポッドにトラフィックをルーティングするタイミングを決定します。

```
readinessProbe:

  exec:

    command:

      - /bin/ls

      - /tmp/cpx_ready

  failureThreshold: 15

  initialDelaySeconds: 60

  periodSeconds: 30

  successThreshold: 1

  timeoutSeconds: 1
```

ライブネスプローブを構成する：実行中にコンテナの健全性を継続的に監視するために、次のスニペットを追加します。実行中にCPXの健全性を監視するために、デプロイメントYAMLでライブネスプローブを定義します。プローブが失敗した場合、Kubernetesはコンテナを再起動して、異常な状態から回復させます。このメカニズムは、応答しないCPXインスタンスを自動的に再作成することで、アプリケーションの可用性を保証します。

```
livenessProbe:

  exec:

    command:

      - /bin/ping

      - -c 1

      - 192.0.0.1

  initialDelaySeconds: 10

  periodSeconds: 5
```

```
failureThreshold: 3
```

```
successThreshold: 1
```

```
timeoutSeconds: 1
```

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、少なくとも以下が必要です。
 - 1 CPU
 - 2 GB RAM

注

NetScaler CPXのパフォーマンスを向上させるために、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4GB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

`docker pull` コマンドを使用して、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、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

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

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 --name <cpx_container_name> <REPOSITORY>:<CPX-
IMAGE_TAG_IN_QUAY>
```

```
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
quay.io/citrix/citrix-k8s-cpx-ingress:14.1-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 パラメーターは必須です。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ピングするように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ピングします。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22を使用
- SNMPには161/UDPを使用します。

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 をデプロイのホストモードで実行している場合、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。

シングルコアまたはマルチコアでブリッジモードでNetScaler CPXを実行するには、特権モードの代わりに **--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。 **--**

`cap-add=NET_ADMIN` オプションを使用する場合は、`docker run` コマンドで `--sysctl kernel.shmmax=1073741824 --sysctl net.ipv6.conf.default.accept_dad=0 --sysctl kernel.core_pattern=/var/crash/core.%e.%p.%s` オプションを使用して、追加のシステム制御設定を手動で行います。これらの追加のシステム制御設定は、非特権モードでは自動的に行われません。

`**--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ており、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 をブリッジまたはなしの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す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には9995
- HTTPSには9996
- SSHには9997
- SNMPには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ネットワークなし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vCPU数と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。
- NetScaler CPXは1～16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、[Citrixライセンスシステムコンポーネント](#) 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ライセンスの種類を指定します。

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イスをNetScaler CPXコンテナに割り当てることができます。空白で区切られたネットワークインターフェイスを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イスは、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イスが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、コンテナへの別のネットワーク接続を構成したり、既存のネットワークを削除したりできます。その後、NetScaler CPXコンテナを再起動して、更新されたネットワークを使用するようにしてください。

```
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信しない場合、アイドル時間中に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フォーマンスを制御できます。`CPX_CONFIG` 環境変数にJSON形式で次の値を指定する必要があります。

- NetScaler CPXコンテナがアイドル状態のときにCPUを解放するようにしたい場合は、`{"YIELD" : "Yes"}` を定義します。
- NetScaler CPXコンテナがアイドル状態のときにCPUを解放せず、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のパラメータです。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、設定ファイル、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たポートは、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、ランタイムメモリ消費が少ないNetScaler CPXの軽量バージョンを提供します。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 リライトおよびレスポ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>
```

次の例では、NetScaler CPX イメージに基づいて軽量コンテナを作成します。

```
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 `newslog` 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展開する際に `newslog` を使用して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 (CPX_CORES=1) のみをサポートします。

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 Compose 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くポート、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記述します。ここで、:

- `<service-name>` は、プロビジョニングするサービスの名称です。
- `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付与します。
- `cap_add` は、NetScaler CPX インスタンスにネットワーク権限を付与します。
- `<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。処理エンジンを追加するごとに、Dockerホストに同等の数のvCPUとGB単位のメモリがあ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一般的に、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

Docker ホストにインストールされている NetScaler CPX インスタンスを管理および監視するには、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用に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するか、プロファイルを作成してから、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 ソフトウェアがインストールされていること。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 インスタンスがインストールされていること。

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. NetScaler Console で新しい CPX インスタンスを追加するには、**追加**をクリックします。
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て、CPX インスタンスを追加できます。
 - 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ます。



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、次に、**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 ConsoleにNetScaler CPXインスタンスを追加するための `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ています。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。

以下は、必要な環境変数とその説明です。

- `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 を作成するための例です。

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 が次の形式で必要です。

```
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。

ConfigMap を作成するには、次のコマンドを使用します。

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

注: ファイル名には `adm_reg_envs` 変数を含める必要があり、パス `/var/admininfo/server/` にマウントする必要があります。

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

```
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ます。Kubernetes環境では、NetScaler CPXは動的に起動したり停止したりできます。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回収するのに数分かかります。NetScaler Consoleサーバーは、NetScaler CPXが停止したときにすぐにライセンスを回収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ライセンスアグリゲーターは、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クラスターにデプロイされたNetScaler CPXライセンスアグリゲーターサービスは、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ライセンスアグリゲーターサービスを使用すると、NetScaler ConsoleサーバーはNetScaler CPXが停止したときにすぐにライセンスを回収できます。

Kubernetesクラスターでは、NetScaler CPXライセンスアグリゲーターサービスは、サイドカーとしてのNetScaler CPXとスタンドアロンデプロイメントの両方をサポートします。

注:

NetScaler CPXライセンスアグリゲーター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ライセンスアグリゲーターは、NetScaler CPXの古いバージョンのライセンス付与をサポートしていません。

NetScaler CPXライセンスアグリゲーターの主な利点

NetScaler CPXライセンスアグリゲーター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デプロイメントのみをサポートできます。NetScaler CPXライセンスアグリゲーター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

アントとして機能できます。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ライセンスアグリゲーターサービスは、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ライセンスアグリゲーターは、ライセンスをNetScaler Consoleサーバーに返却できます。NetScaler CPXライセンスアグリゲーターは、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からライセンスを回収できます。

展開トポロジ

次の図は、Kubernetesクラスター内でのNetScaler CPXライセンスアグリゲーターの展開を示しています。

NetScaler CPXライセンスアグリゲーターの展開(/en-us/cpx/media/cla-deployment.png)

この図では:

- **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 サービスが、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 エージェントの間に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、それらにライセンスを管理します。

Helm チャートを使用した NetScaler CPX License Aggregator の展開

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 データベースのパスワードを指定する必要があります。Kubernetes シークレットを使用してデータベースパスワードを指定でき、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。NetScaler CPX ライセンスのさまざまな種類の詳細については、[NetScaler CPX licensing](#) 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=&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.bandwidthEnterpriseQuantum=&lt;QUANTUM-in-
Mbps>,licenseInfo.bandwidthEnterpriseLowWatermark=&lt;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=&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.bandwidthStandardQuantum=&lt;QUANTUM-
in-Mbps>,licenseInfo.bandwidthStandardLowWatermark=&lt;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クラスターに展開します。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、インストール中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する[HelmチャートGitHubリポジトリ](#)の「**NetScaler CPX License Aggregatorの設定**」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、次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ライセンスアグリゲーターで複数の種類のライセンスを管理する必要がある場合は、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のライセンスにNetScaler CPX License Aggregatorを使用する場合、環境変数 `CLA` をNetScaler CPX展開YAMLで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で以下の環境変数も指定する必要があります。

- `POD_NAME` : ポッドの名前を指定します。ポッドの名前は環境変数としてNetScaler CPXに公開されます。
- `POD_NAMESPACE` : ポッドの名前空間を指定します。ポッド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: 展開例](#)」を参照してください。

NetScaler CPX の構成

Linux Docker ホストを介して CLI プロンプトにアクセスするか、NetScaler NITRO API を使用して、NetScaler CPX インスタンスを構成できます。

コマンドラインインターフェイスを使用して NetScaler CPX インスタンスを構成する

Docker ホストにアクセスし、次の図に示すようにインスタンスの 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。

Linux を使用して CPX を構成する(/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:

Linux を使用して CPX を構成する例(/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 で非デフォルトパスワードを使用するためのサポート

NetScaler CPX は、root アカウントに非デフォルトパスワードを使用することをサポートしています。それは `nsroot` です。NetScaler CPX が展開されると、デフォルト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` と `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` と `nsroot` のユーザーアカウントのデフォルト 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 通信と `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 ファイルシステム内の `/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。実行する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したい場合は、「**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機能を構成すると、Web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、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。ポリシーは、フローレコードが関連付けられたアクションに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。

さらに、AppFlowパラメーターを設定して、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ことができます。各コレクターでは、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。これにより、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）にエクスポートする方法が正確に決定されます。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にNetScaler CPXインスタンスを追加する方法の詳細については、「[NetScaler Consoleを使用したNetScaler CPXインスタンスのインストール](#)」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：

複数のコレクターに同じデータを追加して送信するには、次のコマンドを使用します。

```
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コンテナの展開中に、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、 `#NetScaler Commands` と `#Shell Commands` の2つのタグを含む `.conf` ファイルです。 `#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シナリオが発生した場合、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ものを選択できます。

NetScaler CPXの非デフォルトパスワードについては、[Configuring NetScaler CPX](#)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デプロイされます。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デプロイについては、[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、仮想サーバーは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構成され、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に対して `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。

例えば、次のように構成したい場合：

```
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. 次のコマンドで、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コンテナを構成して、生成されたログをDockerロギングドライバーに転送できます。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構成できます。ロギングドライバーに構成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。

次の例では、NetScaler CPXコンテナは、ロギングドライバーとしてsyslogを使用して、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、次の例では、NetScaler CPXコンテナは、ロギングドライバーとしてSplunkを使用して、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、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**」サブエントリは、ホスト上のホストディレクトリの場所を示します。

マウント元とマウント先の出力例 (/en-us/cpx/media/mount_point_cpx.png)

- 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、`<containerName>` がNetScaler CPXインスタンスの名前である `docker stop <containerName>` コマンドを入力します。

```
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インスタンスで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インスタンスにプライベートIPアドレス（単一IPアドレス）が1つだけ割り当てられます。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。

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。単一IPアドレスまたは127.0.0.1 IPアドレスを使用して、負荷分散（LB）またはコンテンツスイッチング（CS）仮想サーバーを追加します。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。

ワイルドカード負荷分散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
```

East-Westトラフィックフローを有効にするプロキシとしてNetScaler CPXを展開

この展開では、NetScaler CPXインスタンスは、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するプロキシとして機能します。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、通信を行う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、この構成例でNetScaler CPXインスタンスに設定されているIPアドレスとポートを示しています。



このシナリオ例を構成するには、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で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。

ローカライズされた画像(/en-us/cpx/media/cpx-single-host-topology.png)

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プロキシとして配置されます。

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NetScaler CPXコンテナ上で2つのサービスを構成して2つのサーバーを表す必要があります。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構成します。これは、NetScaler CPXが標準のHTTPポート80をNITRO通信用に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ルールを構成する必要はありません。

このシナリオを構成するには、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でのNetScaler CPXの展開

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運用環境に構成できます。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展開されます。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、あるいは物理サーバーのセットをロードバランシングする運用環境では、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生します。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運用環境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ています。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、その要求を2つのサーバーに分散します。

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. NetScaler CPXコンテナで、別のIPアドレスを使用してVIPを構成します。
2. Dockerホストに別の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。

この例の設定では、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。

このシナリオ例を設定するには、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、Linux シェルプロンプトで次のコマンドを実行して設定します。

```
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設定する:

1. NetScaler CPX インスタンスで VIP と 2 つのサービスを設定します。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 ルールを設定します。

このシナリオ例を設定するには、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設定していない場合は、Linux シェルプロンプトで次のコマンドを実行して NAT ルールを設定します。

```
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ロードバランシングできます。

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。サーバーである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。クライアント要求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 IPアドレスと非標準マッピングされた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 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ロードバランシングする本番環境の展開で使用される場合があります。

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1つのサーバー (DB1) が、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要求はNetScaler CPXインスタンスで受信され、その後、5つのサーバーに要求が分散されます。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。クライアント要求がホストのポート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。

例の構成では、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。

この例のシナリオを構成するには、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 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インスタンスをネットワークに直接アクセスす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、その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、その後、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。

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のジョブ機能を使用するか、またはNITRO APIを使用するかして、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
```

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

Kubernetesでは、ConfigMapを使用してNetScaler CPXインスタンスを構成できます。ConfigMap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固有の構成を行うためのすべての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。

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

```
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` は、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で指定された構成で起動すると、`cpx-config` がNetScaler CPXインスタンスに適用されます。

ConfigMap を使用した NetScaler CPX のカスタムメトリクス設定

Kubernetes の NetScaler CPX™ インスタンスに ConfigMap を使用してカスタムメトリクス設定 JSON ファイルを提供できます。これにより、CPX に含まれるデフォルトのメトリクス設定ファイル (`schema.json` および `reference_schema.json`) を上書きすることなく、カスタムメトリクススキーマファイルを追加できます。

起動時に、NetScaler CPX インスタンスはデフォルトのメトリクス設定ファイルを `/var/metrics_conf` ディレクトリに配置します。 `/cpx/custom_metrics_conf` に ConfigMap をマウントすると、CPX は各 JSON ファイルを検証し、有効なファイルをデフォルトとともに `/var/metrics_conf` にコピーします。

注記:

- カスタム JSON ファイルには有効な JSON が含まれている必要があります。無効なファイルは起動時にスキップされ、エラーが `boot.log` にログ記録されます。
- デフォルトファイル (`schema.json` , `reference_schema.json`) は常に最初に配置されます。
- デフォルトファイルと同じ名前のカスタムファイルは、デフォルトファイルを上書きします。

カスタムメトリクスを構成する

ConfigMap を作成し、ConfigMap がマウントされた NetScaler CPX インスタンスを展開することで、NetScaler CPX のカスタムメトリクスを構成できます。

CLI を使用して構成するには

NetScaler CPX インスタンスに提供するカスタムメトリクス設定 JSON ファイルを作成します。以下は、カスタム設定ファイルのサンプルです。

```
{  
  
  "counter": "http_requests_total",  
  
  "label": "method",
```

```
"description": "Total HTTP requests by method"

}
```

`kubectl create configmap` コマンドを使用してファイルから ConfigMap を作成します。たとえば、設定ファイル `custom_schema.json` に基づいて `custom-metrics-conf` という名前の ConfigMap を作成するには:

```
kubectl create configmap custom-metrics-conf --from-  
file=custom_schema.json=./custom_schema.json
```

`kubectl get configmap` コマンドを使用して作成された ConfigMap を表示します。

```
kubectl get configmap custom-metrics-conf -o yaml
```

次の例は、サンプル出力を示しています。

```
apiVersion: v1

data:

  custom_schema.json: |

    {

      "counter": "http_requests_total",

      "label": "method",

      "description": "Total HTTP requests by method"

    }

kind: ConfigMap

metadata:

  creationTimestamp: 2026-05-14T06:26:50Z

  name: custom-metrics-conf
```

```
namespace: default

resourceVersion: "8865149"

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

YAML ファイルを使用して構成するには

NetScaler CPXインスタンスのデプロイに使用するYAMLファイルで、作成したConfigMap

`custom-metrics-conf` を指定します。次のサンプル構成は、デプロイYAMLにConfigMapを含める方法を示しています。

```
apiVersion: v1

kind: Pod

metadata:

  name: cpx-1

  labels:

    app: cpx-daemon

  annotations:

    NETSCALER_AS_APP: "True"

spec:

  containers:

    - name: cpx

      image: "quay.io/netScaler/netScaler-cpx:14.1-72.56"

      securityContext:

        privileged: true

  volumeMounts:
```

```
- name: config-volume

  mountPath: /cpx/bootup_conf

- name: custom-metrics-volume

  mountPath: /cpx/custom_metrics_conf

env:

- name: "EULA"

  value: "yes"

  imagePullPolicy: IfNotPresent

volumes:

- name: config-volume

  configMap:

    name: cpx-config

- name: custom-metrics-volume

  configMap:

    name: custom-metrics-conf
```

NetScaler CPXインスタンスをデプロイします。インスタンスがデプロイされ起動すると、デフォルトのメトリック構成ファイル (`schema.json` および `reference_schema.json`) は `/var/metrics_conf` に配置されます。 `/cpx/custom_metrics_conf` にマウントされたConfigMapからのカスタムJSONファイルは検証され、 `/var/metrics_conf` にコピーされます。

KubernetesノードのローカルDNSキャッシュとしてNetScaler CPXを展開する

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。NetScaler CPXを各KubernetesノードにローカルDNSキャッシュとして展開し、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に転送できるようになりました。これにより、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を展開するには、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。

NetScaler CPXをNodeLocal DNSキャッシュとして展開することをサポートするために、以下の環境変数が導入されました。

- `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クエリを転送します。
- `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。

ノードにNetScaler CPXをDNSキャッシュとして展開する

KubernetesクラスターのローカルDNSキャッシュとしてNetScaler CPXを展開するには、以下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する。

ワーカーノードで:

- DNSリクエストをNetScaler CPXに転送するために、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マスターノードで以下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` ファイルは以下のとおりです。

```
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ための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。このステップでは、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` が作成されます。

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

以下は、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
```

以下はサンプルConfigMap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` ファイルは以下のとおりです。

```
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、ワーカーノードで以下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構成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、以下の形式で `--cluster-dns` 引数の値を変更します。

```
--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構成を編集し、`--cluster-dns` 引数に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し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。

構成手順

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を設定し、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
dHM7ZTqlA1tHIG+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/EMo7iQIcBBABCgAGBQJXSWBlAAoJEFyzYeVS
```

```
\+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通信を設定する

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は同じイメージです。ただし、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構成し、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 で負荷分散を構成して 2 つの Web サーバーの負荷を分散し、構成をテストします。

NGINX Web サーバーをインストールする

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

```
$ 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

$
```

負荷分散のために NetScaler CPX を構成し、両方の Web サービスへの負荷の分散を確認する

```
$ **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:~\# **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

RHISTate: 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

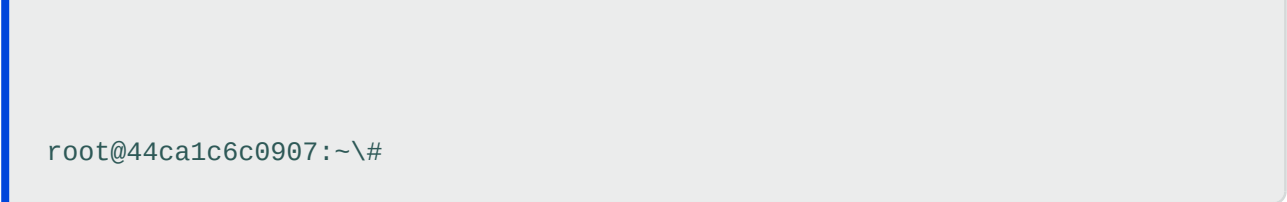
Current client connections	--	0
Current Client Est connections	--	0
Current server connections	--	0
Current Persistence Sessions	--	0
Requests in surge queue	--	0
Requests in vserver's surgeQ	--	0
Requests in service's surgeQs	--	0
Spill Over Threshold	--	0
Spill Over Hits	--	0
Labeled Connection	--	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	172.17.0.3	80	HTTP	UP	51
0/s					

web2	172.17.0.4	80	HTTP	UP	50
0/s					

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回避策を適用できます。

- NetScaler CPX でメモリ使用率が常に高いのはなぜですか？

メモリ使用率は、NetScaler CPX の現在のメモリ使用量と、NetScaler CPX がオペレーティングシステム (OS) から自身に割り当てる合計メモリを比較することによって決定されます。

NetScaler CPX は、起動と基本的な操作のために、OS から必要最小限のメモリを割り当てることから開始します。当初、NetScaler CPX はその操作に必要な最小限のメモリを割り当てるため、メモリ消費率は高くなります。最終的に、NetScaler CPX の操作の拡張に伴い、より多くのリソースが必要になると、NetScaler CPX は OS から追加のメモリを段階的に割り当て、メモリ使用量も高くなり、結果としてメモリ使用率が高くなります。

したがって、NetScaler CPX の動的なメモリ割り当て戦略により、NetScaler CPX は通常、高いメモリ使用率を示しますが、これは利用可能なメモリの不足ではなく、リソースの効率的な使用を反映しています。

- 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 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
```

次は、`tty:true` オプションを使用した NetScaler CPX ポッド展開の例です。

```
containers:

  - name: cpx-ingress

    image: "quay.io/citrix/citrix-k8s-cpx-ingress:13.0-58.30"

    tty: true

    securityContext:

      privileged: true

    env:
```

NetScaler CPX ファイルシステムの `/cpx/log/boot.log` ファイルで、より多くのブートログを見つけることができます。

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` ディレクトリで確認できます。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 pod 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 pod のステータスを確認できます。pod のステータスを確認するには、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
```

pod イベントでコンテナが起動していると表示される場合は、pod のログを確認する必要があります。

- NetScaler CPX pod 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？

docker のボリュームマウント機能を使用して、`/cpx` ディレクトリをホストのファイルシステムにマウント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 pod と Kubernetes マスターノード間でファイルをコピーするには、次のいずれかのコマンドを使用できます。

kubectl cp: pod 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 pod と Kubernetes ノード間でファイルをコピーできます。pod 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イスを起動します。NetScaler CPX pod のシェルインターフェイスを起動するには、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"
```

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*ファイルでキャプチャされたパケットを表示できます。

- NetScaler CPXが `LS_IP=<ADM-IP>` 環境変数で展開されているのに、ライセンスサーバーが構成されてい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。NetScaler CPXノードからNetScaler Consoleへの接続を確認するには、`ping <ADM-IP>` コマンドを使用できます。

NetScaler Consoleがノードからアクセスできる場合は、`/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てください。

```
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。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するには、`kubectl describe` コマンドを実行できます。

```
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"}'` 環境変数とともに展開されている場合、トラフィックがないか少ない場合でも、NSPPEプロセスはCPU使用率を100パーセント消費します。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使用率を占有または消費しないように構成されています。

- NetScaler Consoleへの登録に必要な環境変数とともに展開されたにもかかわらず、NetScaler CPXがNetScaler Consoleに表示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`*ファイルにあります。

NetScaler CPXポッドから `ping` コマンドを使用して、NetScaler Consoleの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コンテナに構成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？

コマンド `cli_script.sh` 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.