NetScaler CPX 14.1

Contents

NetScaler CPX について

November 23, 2023

NetScaler CPX は、Docker ホストで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配信コントローラーです。NetScaler CPX を使用することにより、Docker エンジン機能を利用し、NetScaler の負荷分散機能とトラフィック管理機能を、コンテナベースのアプリケーション向けに活用できます。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 Docker ホストで展開できます。

NetScaler CPX インスタンスの最大スループットは 1Gbps です。

NetScaler のコンテナ化されたフォームファクターである NetScaler CPX は、Kubernetes 環境にうまく統合でき、NetScaler クラウドネイティブソリューションに欠かせない要素となっています。NetScaler のクラウドネイティブソリューションを使用すると、Kubernetes 環境でスピード、俊敏性、効率性を備えたソフトウェアアプリケーションを作成および配信できます。NetScaler の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。
詳しくは、「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、ユーザーが Docker とその機能を理解していることを前提としています。Docker について詳しくは、Docker のドキュメント（https://docs.docker.com）を参照してください。

サポートされる機能

NetScaler CPX では、次の機能がサポートされます。

- アプリケーションの可用性
  - L4 の負荷分散および L7 のコンテンツスイッチ
  - SSL オフロード
  - IPv6 プロトコル変換
  - Microsoft SQL、MySQL の負荷分散
  - AppExpert レートコントロール
  - 利用者に応じたトラフィックステアリング
  - サージ保護と優先度によるキューイング
  - 動的ルーティングプロトコル
- アプリケーションの速度向上
  - クライアント TCP とサーバー TCP 最適化
  - キャッシュリダイレクト
  - AppCompress
  - AppCache
- アプリケーションのセキュリティ

  - L7 の書き換えおよびレスポンダー
  - L4 DoS 攻撃防御
  - L7 DoS 攻撃防御
  - 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 機能について詳しくは、「NetScaler WebApp Firewall」を参照してください。
  - アプリケーショントラフィックの認証、承認、監査（AAA）

- TCP プロトコルの最適化
  - マルチパス TCP
  - Binary Increase Congestion Control（BIC）および CUBIC TCP

- 簡単な管理機能
  - Web ログ
  - AppFlow
  - NetScaler Application Delivery Management
  - アクション分析

- アプリケーションの最適化
  - 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）
- 高可用性（レイヤー 2 とレイヤー 3 の両方）

> 注:
>
> NetScaler CPX アプライアンスに割り当てられたインターフェイス（Linux ホスト）では、Rx、Tx、GRO、GSO、LRO などのインターフェイス機能は無効になっています。これらの機能は、NetScaler CPX アプライアンスを停止した後も無効状態のままです。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。

サポートされるプラットフォーム

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

- Kubernetes
- Red Hat OpenShift
- パブリッククラウド
  - Amazon Elastic Kubernetes Service（EKS）
  - Azure Kubernetes Service（AKS）
  - グーグル Kubernetes エンジン (GKE)

- Rancher
- Pivotal Container Service（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

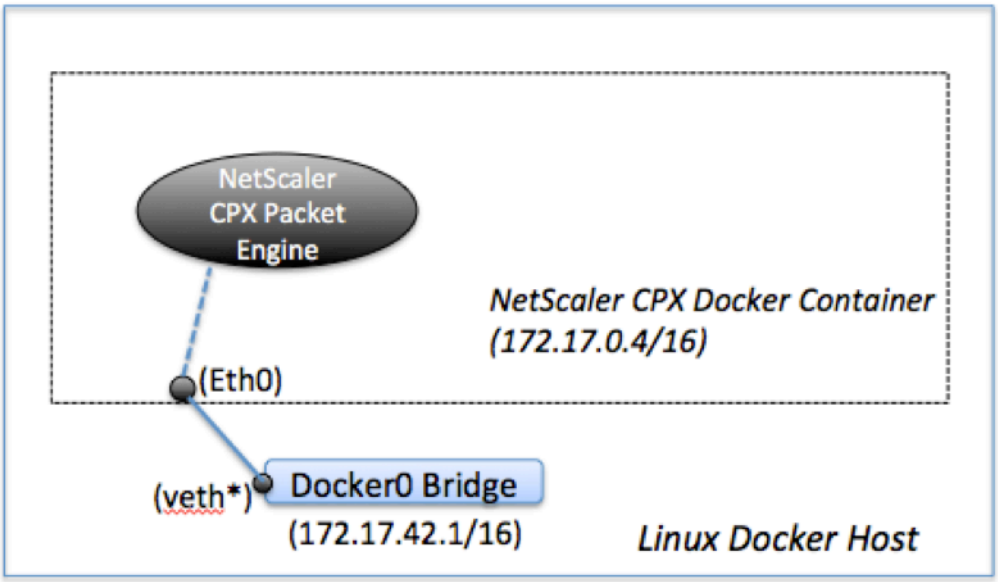
November 23, 2023

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。

NetScaler CPX インスタンスを Docker ホストにプロビジョニングすると、Docker エンジンによって仮想インターフェイス eth0 が CPX インスタンスに作成されます。この eth0 インターフェイスは、docker0 ブリッジで仮想インターフェイス（veth*）に直接接続されます。また、Docker エンジンによって、ネットワーク 172.17.0.0/16 の NetScaler CPX インスタンスに IP アドレスが割り当てられます。

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 アドレスです。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 ブリッジからのすべての受信トラフィックが、NetScaler CPX インスタンスの eth0 インターフェイスで受信され、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。



単一 **IP** アドレスが **NetScaler CPX** で機能するしくみ

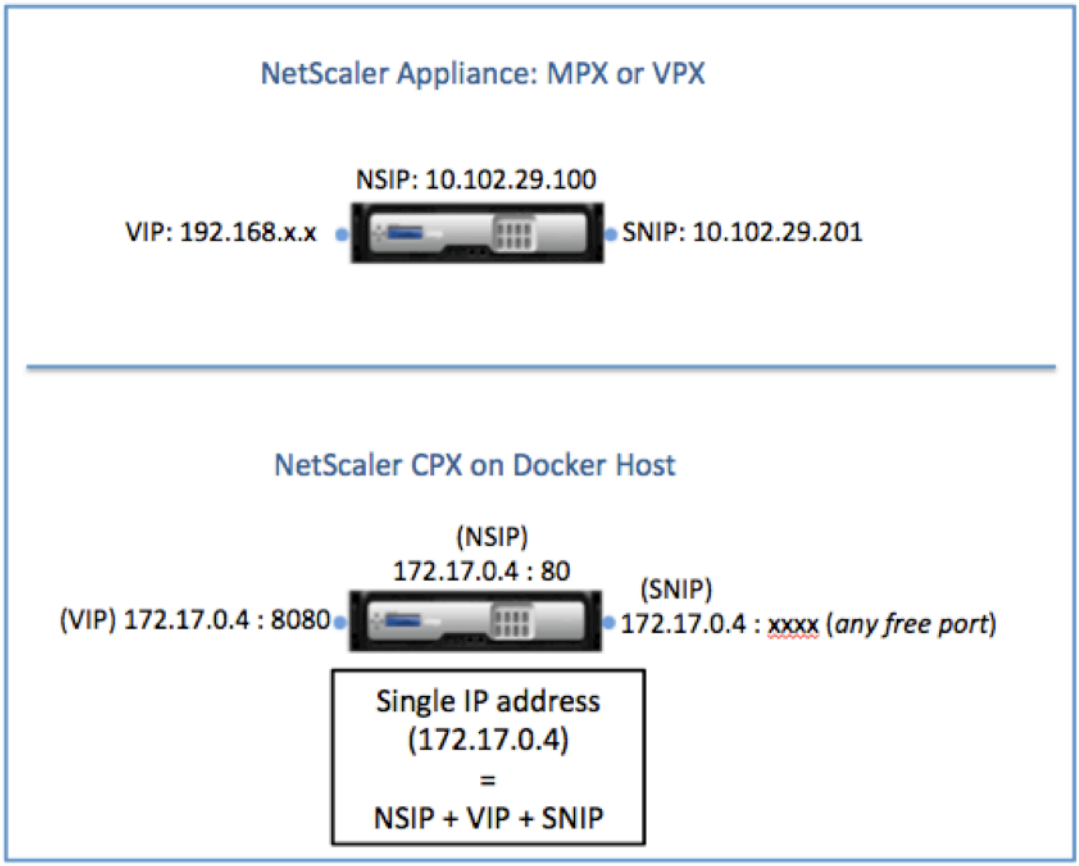
通常の NetScaler MPX または VPX アプライアンスでは、少なくとも以下に示す 3 つの IP アドレスが機能している必要があります:

- 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やり取りするためのサブネット IP（SNIP）アドレス
- クライアント要求を受け付ける仮想サーバー 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用に使用される、単一 IP アドレスで動作します。

プロビジョニング中は、Docker エンジンによって、1 つのプライベート IP アドレス（単一 IP アドレス）のみが NetScaler CPX インスタンスに割り当てられます。NetScaler インスタンスの 3 つの IP 機能が 1 つの IP アドレス上で多重化されます。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。

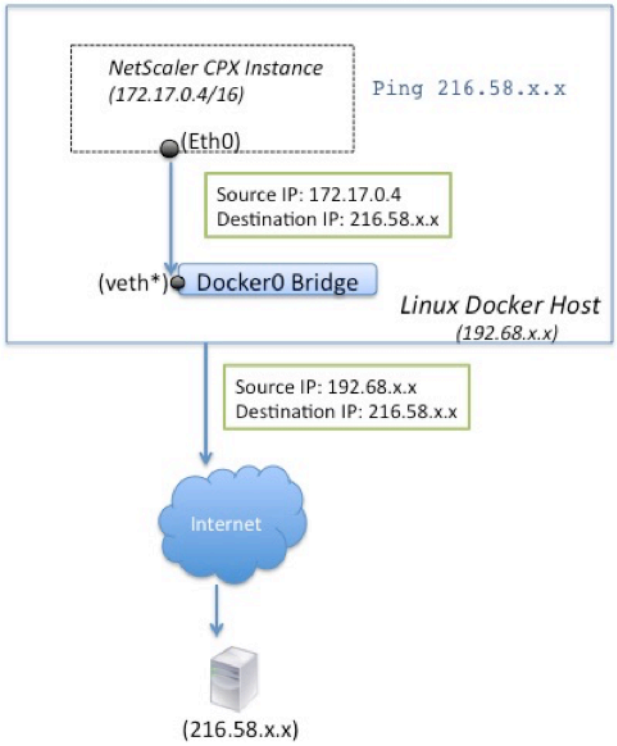
次の図は、単一 IP アドレスを使用して NSIP、SNIP、および VIP の機能を実行する方法を示しています。



NetScaler CPX インスタンスからの要求のトラフィックフロー

Docker では、NetScaler CPX インスタンスからのトラフィックが docker0 の IP アドレスに転送されるよう、IP の一覧と NAT 規則が暗黙的に構成されます。

次の図は、NetScaler CPX インスタンスからの ping 要求が、どのようにして宛先に到達するかを示しています。



この例の ping 要求は、ソース IP アドレスを NetScaler CPX IP アドレス（172.17.0.4）として、eth0 インターフェイスのパケットエンジンによって送信されます。次に、Docker ホストが実行するネットワークアドレス変換（NAT）によって、ホスト IP アドレス（192.68.x.x）がソース IP アドレスとして追加され、要求が宛先（216.58.x.x）に送信されます。ターゲット IP アドレスからの応答は、逆の順序で同じパスをたどります。Docker ホストが応答に対して NAT を実行し、eth0 インターフェイスの NetScaler CPX インスタンスに応答を転送します。

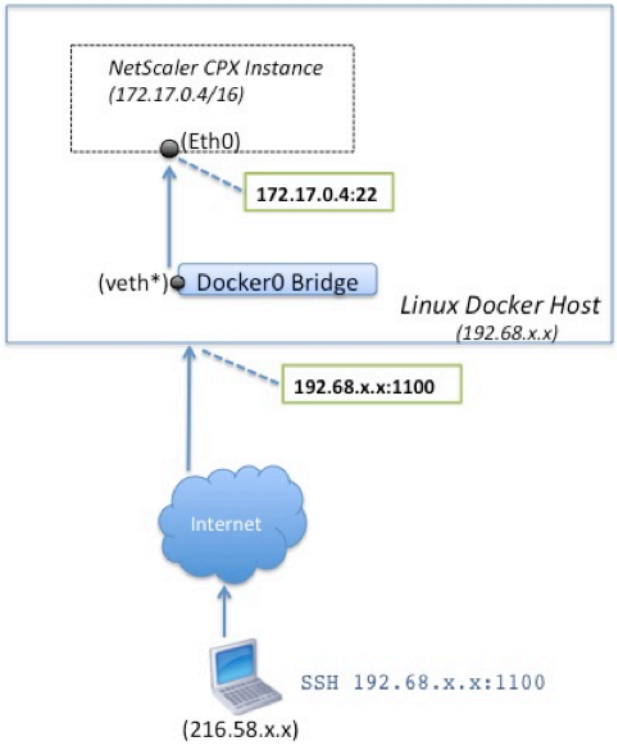
外部ネットワークからの要求のトラフィックフロー

NetScaler CPX のプロビジョニング中に外部通信を可能にするには、Docker で 80 や 22 などの特定のポートや、その他の任意のポートが公開され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されるように設定されているポートがない場合は、Docker ホストに NAT 規則を構成して、これらのポートを使用可能にする必要があります。

Docker ホストは、インターネットからのクライアント要求を受信し、次にポートアドレス変換（PAT）を実行して、パブリック IP アドレスおよびポートを NetScaler CPX インスタンスの単一 IP アドレスとポートにマップし、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、トラフィックを NetScaler CPX の単一 IP アドレスおよびポートに転送する方法を示しています。



この例で、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 ホストのポート 1100 にマップされます。クライアントの SSH 要求は、IP アドレス 192.68.x.x のポート 1100 で受信されます。Docker ホストが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 の単一 IP アドレス 172.17.0.4 にマップし、クライアント要求を転送します。

NetScaler CPX ライセンス

March 25, 2024

NetScaler CPX は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 、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ます。アプリケーション配信のパフォーマンスを向上させるには、ライセンスが適用された CPX が必要です。NetScaler CPX はプールライセンスをサポートしています。NetScaler ADM は、NetScaler CPX インスタンスのライセンスを取得するライセンスサーバーとして機能します。

NetScaler ADM は、オンプレミスとクラウドサービスの両方でも利用できます。NetScaler ADM 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ライセンスを管理できます。

NetScaler ADM オンプレミスについて詳しくは、「NetScaler ADM オンプレミス」を参照してください。NetScaler ADM サービスについて詳しくは、「NetScaler ADM サービス」を参照してください。

NetScaler CPX ライセンスの種類

NetScaler CPX は、オンプレミスおよびクラウドベースの展開の帯域幅と仮想 CPU（コア）プールのライセンスをサポートしています。

帯域幅プール:NetScaler CPX ライセンスは、インスタンスの帯域幅消費量に基づいて割り当てることができます。プールされたライセンスを使用して、インスタンスに必要な帯域幅をその要件を超えないように割り当てることで、帯域幅の使用率を最大化できます。現在、NetScaler CPX はプレミアム帯域幅プールライセンスのみをサポートしています。

vCPU プール:CPU 使用量ベースの仮想ライセンスでは、ライセンスによって特定の NetScaler CPX インスタンスに割り当てられる CPU の数が指定されます。そのため、NetScaler CPX は、ライセンスサーバーから特定の数の仮想 CPU のみのライセンスをチェックアウトできます。NetScaler CPX は、システムで実行されている CPU の数に応じてライセンスをチェックアウトします。vCPU プールの詳細については、「NetScaler 仮想 CPU ライセンス」を参照してください。

NetScaler CPX インスタンスでサポートされるプール容量

Product	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：これは CPU の周波数、世代などによって異なります。	20Mbps	1	16	10Mbps

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 の使用量ベースまたは成長に応じた支払いベースのライセンスモデルを開発中です。準備が整い次第、パブリッククラウド市場で使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 プール容量: NetScaler CPX インスタンスが 1 つのインスタンスライセンスをチェックアウトし、必要な帯域幅のみをチェックアウトできる共通のライセンスプール。インスタンスでこれらのリソースが不要になっ

た場合、インスタンスはリソースを共通プールにチェックインし、これらのライセンスを必要とする他のインスタンスが利用できるようになります。

NetScaler CPX チェックインおよびチェックアウトライセンス： NetScaler ADM は、NetScaler CPX インスタンスのライセンスをオンデマンドで割り当てます。NetScaler CPX インスタンスは、NetScaler CPX インスタンスがプロビジョニングされたときに NetScaler ADM からライセンスをチェックアウトし、インスタンスが破棄されたときにライセンスを NetScaler ADM にチェックインし直すことができます。

NetScaler CPX の動作**:** 1 つの NetScaler CPX インスタンスが最大 1 Gbps のスループットをチェックアウトし、帯域幅ライセンスプールからはチェックアウトせず、インスタンスプールからのみチェックアウトします。NetScaler CPX は、最大 1 Gbps の帯域幅使用率までこのように動作します。たとえば、CPX インスタンスが 200Mbps 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 エクスプレス

NetScaler CPX Express は、オンプレミスとクラウドのどちらにも無料で導入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 CPX インスタンスに適用されます。NetScaler CPX のすべての機能は、NetScaler CPX Express ライセンスで利用できます。CPX でサポートされている機能について詳しくは、「NetScaler CPX について」を参照してください。NetScaler コミュニティサポートは NetScaler CPX Express では利用できますが、エンタープライズサポートではご利用いただけません。

NetScaler CPX Express ライセンスは、100Mbps の帯域幅容量を提供します。大容量、本番環境への導入、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を取得する必要があります。

> 注:
>
> NetScaler CPX バージョン 14.1 ビルド 14.1-17.38 以降、CPX Express ライセンスの帯域幅は 20Mbps から 100Mbps 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。

NetScaler CPX ライセンスモデル

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

自社の要件に基づいて、次のモデルのいずれかを選択できます:

- ADM サービスによる NetScaler CPX の帯域幅ベースのライセンス
- ADM サービスによる NetScaler CPX の vCPU ベースのライセンス

- ADM オンプレミスからの NetScaler CPX の帯域幅ベースのライセンス
- オンプレミス ADM による NetScaler CPX の vCPU ベースのライセンス

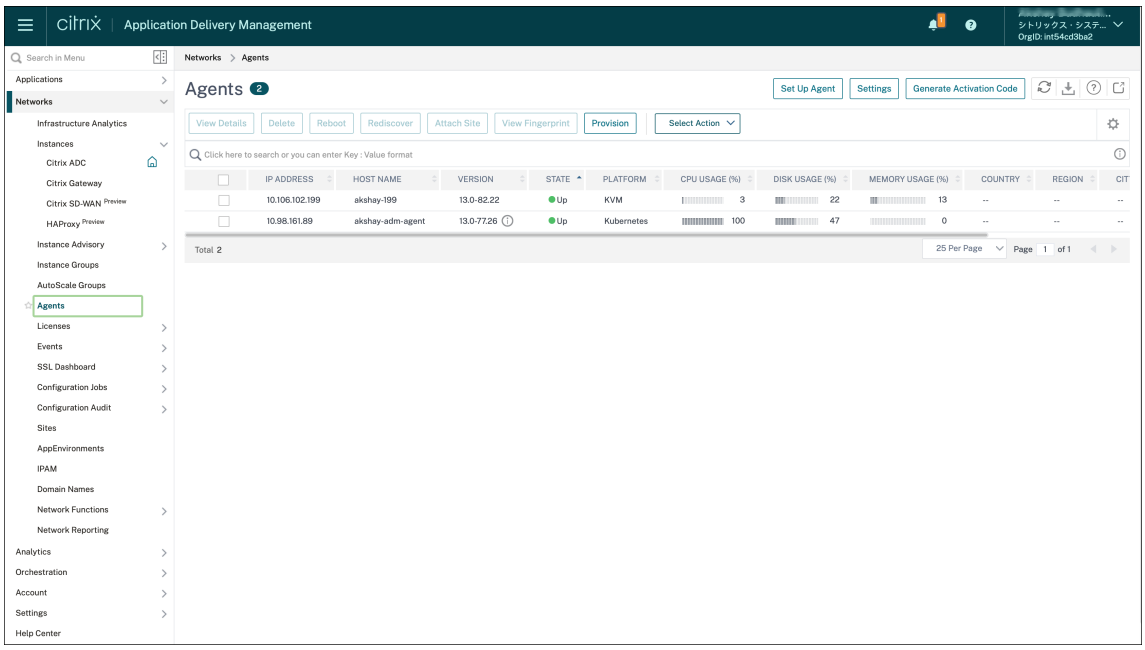
NetScaler CPX 用の **NetScaler ADM** サービスから帯域幅ベースおよび **vCPU** ベースのライセンスをプロビジョニングします

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

1. NetScaler ADM をセットアップします。

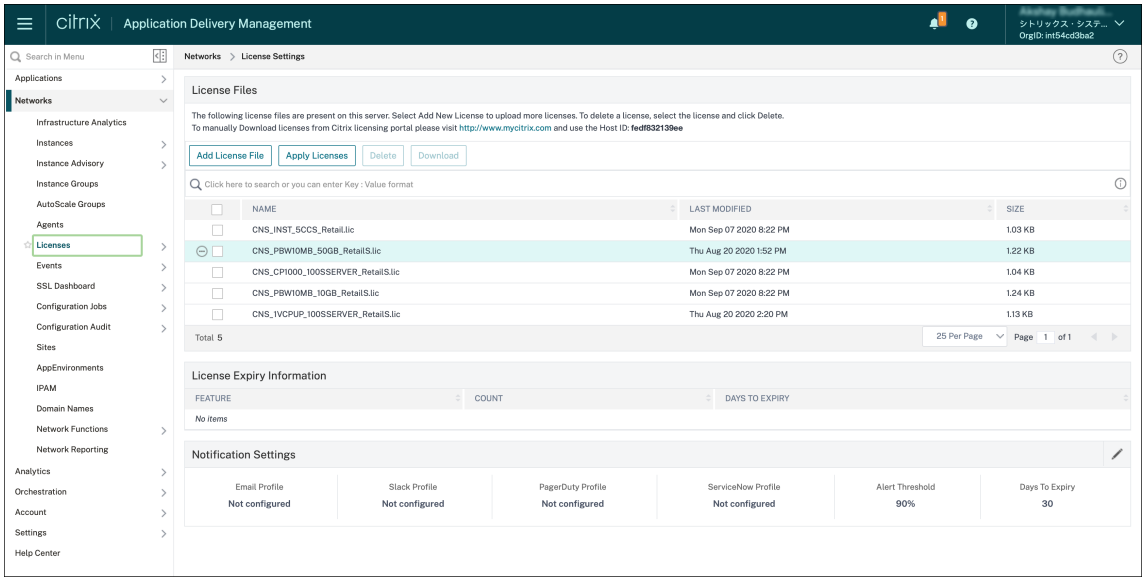
   NetScaler ADM サービスのセットアップが NetScaler ADM エージェントで動作していることを確認します。NetScaler CPX ライセンスが機能するには、NetScaler ADM サービスと NetScaler ADM エージェントアカウントが必要です。NetScaler ADM サービスと NetScaler ADM エージェントの設定について詳しくは、「NetScaler ADM サービス」を参照してください。

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



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

   ADM サービスに使用できる帯域幅ライセンスのプールがあることを前提としています。NetScaler ADM へのライセンスファイルのアップロードについて詳しくは、「プール容量の構成」を参照してください。次の図では、`CNS_INST_200CC_Retail.lic`は帯域幅とインスタンスのライセンスプールとして使用されています。



3. Kubernetes クラスターに NetScaler CPX インスタンスをデプロイします。NetScaler CPX インスタンスのライセンスを取得するには、次の環境変数が NetScaler CPX YAML ファイルに追加されていることを確認してください。

   NetScaler ADM サービスからの帯域幅ベースのライセンスでは、YAML ファイルに次の環境変数を指定します:

   - name: “LS_IP”
     value: “10.105.158.166” //手順 1 の ADM エージェント IP
   - name: “LS_PORT”
     value: “27000” //ADM ライセンスサーバーがリスンするポート
   - name: “BANDWIDTH”
     value: “3000” //CPX に割り当てる Mbps 単位の容量
   - name: “EDITION”
     value: “Standard” or “Enterprise” //Standard、Platinum、および Enterprise を含む特定のライセンスエディションを選択します。デフォルトでは、Platinum が選択されています。

   NetScaler ADM サービスから vCPU ベースのライセンスを取得する場合は、YAML ファイルに次の環境変数を指定します:

   - name: “LS_IP”
     value: “10.102.216.173” //手順 1 の ADM エージェント IP
   - name: “LS_PORT”
     value: “27000” //ADM ライセンスサーバーがリスンするポート
   - name: “CPX_CORES”
     value: “4” //割り当てるコア数
   - name: “PLATFORM”
     value: “CP1000” //コアの数。チェックアウトする数はコアの数と同じです。

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

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

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

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

6. 次のコマンドを使用して NetScaler CPX にログインし、インスタンス情報を確認します:

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

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

```
cli_script.sh “show licenseserver”
cli_script.sh “show capacity”
```

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

NetScaler ADM をオンプレミスから **NetScaler CPX** に帯域幅ベースのライセンスと **vCPU** ベースのライセンスをプロビジョニングします

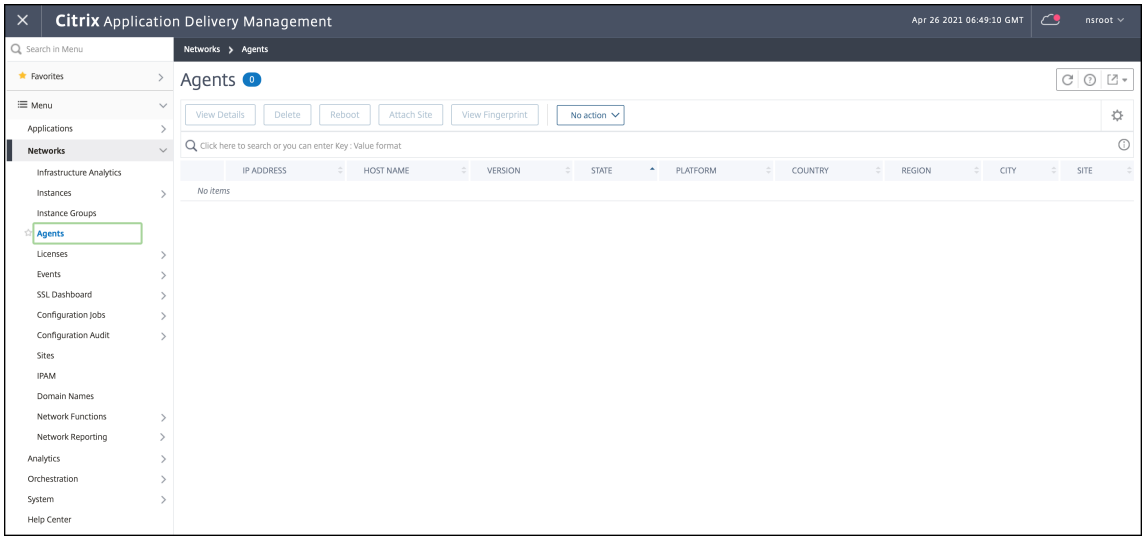
次の手順を実行して、オンプレミスの NetScaler ADM から NetScaler CPX に帯域幅ベースと vCPU ベースをプロビジョニングします。

1. NetScaler ADM をセットアップします。

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

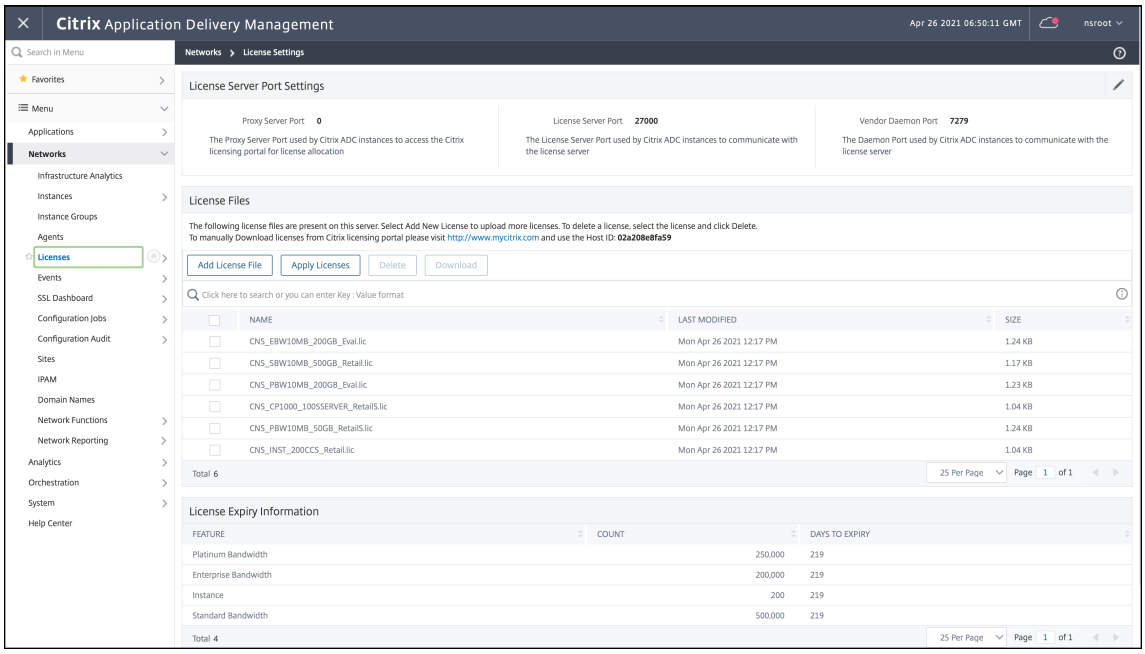
   NetScaler ADM オンプレミスおよび NetScaler ADM エージェントの設定について詳しくは、「NetScaler ADM サービス」を参照してください。

   注：この例では、ADM オンプレミスとともに組み込みの ADM エージェントが使用されています。次の画像では、エージェントが展開されていないことがわかります。

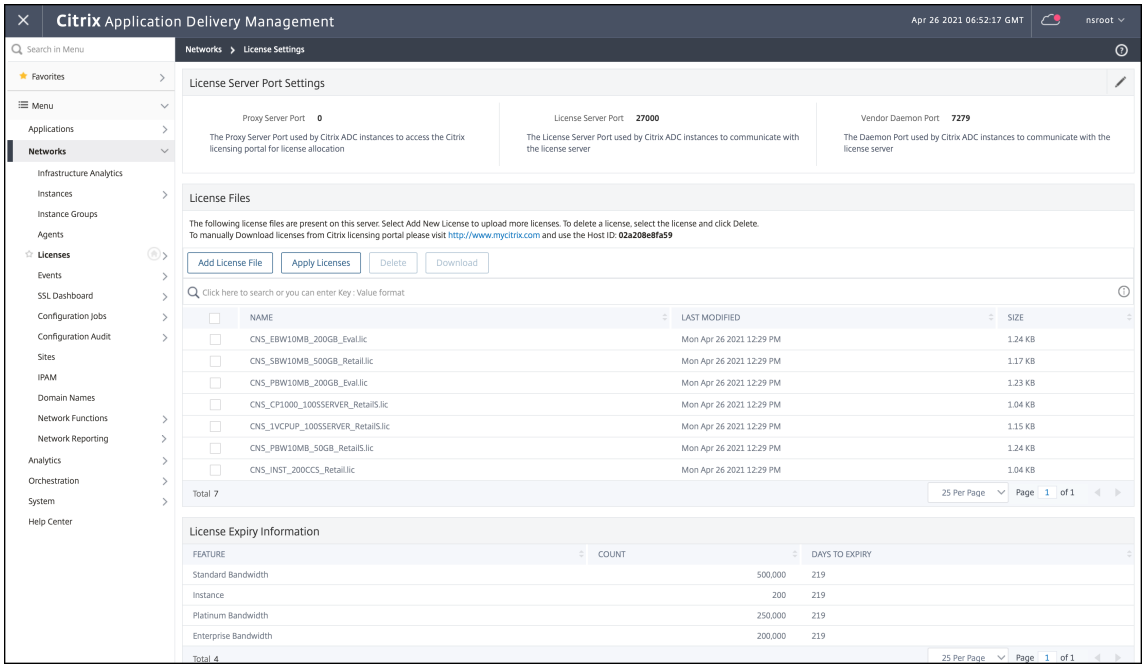


2. NetScaler インスタンスのライセンスプールをオンプレミスの ADM に追加します。

   ADM オンプレミスに使用できる帯域幅ライセンスのプールがあることを前提としています。NetScaler ADM へのライセンスファイルのアップロードについて詳しくは、「ライセンス」を参照してください。次の図では、`CNS_INST_200CC_Retail.lic`は帯域幅とインスタンスのライセンスプールとして使用されています。



   次の図では、CP1000 が vCPU ライセンスプールとして使用されています。



3. Kubernetes クラスターに NetScaler CPX インスタンスをデプロイします。NetScaler CPX インスタンスのライセンスを取得するには、次の環境変数が NetScaler CPX YAML ファイルに追加されていることを確認してください。

   オンプレミスの NetScaler ADM から帯域幅ベースのライセンスを取得する場合は、YAML ファイルに次の環境変数を指定します:

   - name: “LS_IP”
     value: “10.105.158.144”// ADM オンプレミスインスタンス IP。ADM エージェントが展開されている場合、これが手順 1 で説明したエージェント IP アドレス
   - name: “LS_PORT”
     value: “27000”//ADM ライセンスサーバーがリスンするポート
   - name: “BANDWIDTH”
     value: “3000”//CPX に割り当てる Mbps 単位の容量

   オンプレミスの NetScaler ADM から vCPU ベースのライセンスを取得する場合は、YAML ファイルに次の環境変数を指定します:

   - name: “LS_IP”
     value: “10.105.158.144”// ADM オンプレミスインスタンス IP。ADM エージェントが展開されている場合、これが手順 1 で説明したエージェント IP
   - name: “LS_PORT”
     value: “27000”//ADM ライセンスサーバーがリスンするポート
   - name: “CPX_CORES”
     value: “4”// 割り当てるコア数

- name: “PLATFORM”
  value: “CP1000” //コアの数。チェックアウトする数はコアの数と同じです。

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

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

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

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

6. 次のコマンドを使用して NetScaler CPX にログインし、インスタンス情報を確認します:

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

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

```
cli_script.sh “show licenseserver”
cli_script.sh “show capacity”
```

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

展開をクリーンアップするためのコマンド

次のコマンドを使用して、さまざまな YAML 展開をクリーンアップできます:

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

Docker での NetScaler CPX インスタンスの展開

March 25, 2024

NetScaler CPX インスタンスは、Quay コンテナレジストリの Docker イメージファイルとして利用できます。インスタンスをデプロイするには、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デプロイします。

前提条件

以下の点について確認してください:

- Docker ホストシステムが少なくとも次のものを備えていること:
  - CPU 1 つ
  - 2GB の RAM

    > 注: NetScaler CPX のパフォーマンスを向上させるために、NetScaler CPX インスタンスを起動する処理エンジンの数を定義できます。処理エンジンを追加するごとに、Docker ホストにエンジンと同じ数の vCPU と GB のメモリが備えら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は、Docker ホストは 4 つの vCPU と 4GB のメモリを備えている必要があります。

- Docker ホストシステムが Linux Ubuntu のバージョン 14.04 以降を実行している。
- Docker バージョン 1.12 が Linux ホストシステムにインストールされている。Linux での Docker のインストールについて詳しくは、Docker のドキュメントを参照してください。
- Docker ホストからインターネットに接続できる。

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

  注: 以下の kubelet および kube-proxy バージョンにはセキュリティ上の脆弱性があるため、これらのバージョンで Citric NetScaler CPX を使用することは推奨されません:

  - 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、Mitigate this vulnerability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 コンテナーレジストリから NetScaler CPX イメージをダウンロードし、環境にデプロイできます。次の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ます:

```
1    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:

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


次のコマンドを使用して、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します:

```
root@ubuntu:~# docker images | grep 'citrix-k8s-cpx-ingress'
quay.io/citrix/citrix-k8s-cpx-ingress                    13.0-64.35
                952a04e73101        2 months ago        469 MB
```

Quay コンテナレジストリから最新の NetScalerCPX イメージを展開できます。

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

ホストにロードした NetScaler CPX Docker イメージを使用して、ホスト上で 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、NetScaler CPX インスタンスをデフォルトの NetScaler CPX 構成でインストールします。

> 重要:
>
> CPX Expressから NetScaler CPX Express をダウンロードした場合は、CPX Expressで入手できるエンドユーザー使用許諾契約（EULA）を読んで理解し、NetScalerCPX インスタンスを展開する際に EULA に同意するようにしてください。

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

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

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

この例では、NetScaler CPX`mycpx`Docker イメージに基づいて名前の付いたコンテナを作成します。

`-P`パラメーターは必ず指定する必要があります。NetScaler CPX Docker イメージによってコンテナ内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、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することになります。このマッピングは競合を防ぐために行われます。あとで同じ Docker ホストに複数の NetScaler CPX コンテナを作成する場合。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するごとに設定されます。ポートは次のように使用されます:

- 9080 は HTTP 用

- 9443 は HTTPS 用
- 22 は SSH 用
- 161/UDP は SNMP 用。

静的なポートマッピングを行う場合は、-p パラメーターを使用して手動でポートを設定します。

--privileged=**true**オプションは、コンテナを特権モードで実行するために使用されます。NetScaler CPX をホストモードのデプロイメントで実行している場合は、NetScaler CPX にすべてのシステム権限を与える必要があります。

NetScaler CPX を単一コアまたは複数コアのブリッジモードで実行するには、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権限で実行できます。--cap-add=NET_ADMINオプションでは、docker run コマンドの--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は標準的な docker run コマンドオプションで、コンテナがホストネットワークスタックで実行され、すべてのネットワークデバイスへのアクセス権を持つよう指定します。

> 注
>
> NetScaler CPX をブリッジまたはネットワークなしで実行している場合は、このオプションは無視してください。

-e NS_NETMODE="HOST" は NetScaler CPX 固有の環境変数で、これを使用して NetScaler CPX をホストモードで起動するように指定できます。NetScaler CPX がホストモードで起動すると、NetScaler CPX への管理アクセス用に、ホストマシン上で 4 つのデフォルト iptables ルールを構成します。この場合は次のポートを使用します:

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

異なるポートを指定する場合は、次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、NetScaler CPX 固有のオプションの環境変数です。このオプションを使用して、NetScaler CPX コンテナを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改善できます。

注：NetScaler CPX は、1 コアから 16 コアまでサポートできます。

> 注
>
> 処理エンジンを追加するごとに、Docker ホストにエンジンと同じ数の vCPU と GB のメモリが備えら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は、Docker ホストは 4 つの vCPU と 4GB のメモリを備えている必要があります。

`-e EULA = yes`は NetScaler CPX 固有の必須環境変数であり、CPX Expressで入手可能なエンドユーザー使用許諾契約（EULA）を読んで理解していることを確認するために必要です。

`-e PLATFORM=CP1000`パラメーターは、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は、`-e CPX_NW_DEV`環境変数を使用して NetScaler CPX コンテナに専用のネットワークインターフェイスを割り当てることができます。ネットワークインターフェイスはスペースで区切って定義する必要があります。定義したネットワークインターフェイスは、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際に、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は、コンテナへの別のネットワーク接続を設定したり、既存のネットワークを削除したりするなど、コンテナネットワークを変更できます。次に、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。

```
1 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
2 <!--NeedCopy-->
```

`-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可能にする NetScaler CPX 固有の環境変数です。NetScaler CPX が処理すべき受信トラフィックをまったく受け取らないときは、このアイドル時間中に CPU が解放されてスループットパフォーマンスが低下します。このような場合は、`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`環境変数には、JSON 形式で次のような値を入れる必要があります：

- NetScaler CPX コンテナでアイドル時に CPU を解放する場合は、次のように定義します。`{ "YIELD" : “Yes” }`

- NetScaler CPX コンテナがアイドル状態の CPU の解放を回避して高スループットのパフォーマンスを得ることができるようにする場合は、次のように定義します。{ “YIELD” : “No” }

```
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
<!--NeedCopy-->
```

```
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
<!--NeedCopy-->
```

-vパラメーターは、NetScaler CPX マウントディレクトリ/cpxのマウントポイントを指定するオプションパラメーターです。マウントポイントはホスト上のディレクトリであり、そこに/cpxディレクトリをマウントします。/cpxディレクトリには、ログ、構成ファイル、SSL 証明書、コアダンプファイルが保存されています。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所有している場合は、そのライセンスをライセンスサーバーにアップロードして、docker run コマンドを-e LS_IP=<LS_IP_ADDRESS> -e LS_PORT=<LS_PORT>パラメーターを使用してライセンスサーバーの場所を指定できます。この場合は、EULA を承認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
<!--NeedCopy-->
```

各項目の意味は次のとおりです:

- 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。

次のコマンドを使用すると、システムで実行されているイメージと、標準ポートにマップされたポートを表示できます: docker ps

docker run コマンドを使用してより軽量なバージョンの **NetScaler CPX** をデプロイする

NetScaler は NetScaler CPX の軽量版を提供しており、ランタイムメモリの消費量が少なくて済みます。NetScaler CPX の軽量バージョンは、サービスメッシュ展開のサイドカーとして導入できます。

NetScaler CPX の軽量バージョンは、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 の負荷分散および L7 のコンテンツスイッチ
  - SSL オフロード
  - IPv6 プロトコル変換

- アプリケーションのセキュリティ
  - L7 の書き換えおよびレスポンダー

- 簡単な管理機能
  - Web ログ
  - 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>
<!--NeedCopy-->
```

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
<!--NeedCopy-->
```

NetScaler CPX の軽量バージョンでは、デフォルトでnewnslogを使用したロギングは無効になっています。これを有効にするには、より軽量なバージョンの NetScaler CPX を起動するときに、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しているときにnewnslogを使用してロ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>
<!--NeedCopy-->
```

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

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

Docker Compose ツールを使用して、単一または複数の 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、最初に構成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 インスタンスの権限を指定します。

> 重要
>
> Docker Compose ツールをホストにインストールし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. 次の場所に Compose ファイルを作成します:

- **<service-name>** は、プロビジョニングするサービスの名前です。
- **image:<repository>:<tag>** は、NetScaler CPX イメージのリポジトリとバージョンを示します。
- **privileged: true** は、NetScaler CPX インスタンスのすべての root 特権を提供します。
- **cap_add** は、NetScaler CPX インスタンスにネットワーク特権を提供します。
- **<host_directory_path>** は、NetScaler CPX インスタンスをマウントする Docker ホストのディレクトリを指定します。
- **<number_processing_engine>** は、NetScaler CPX インスタンスを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 ホストにエンジンと同じ数の vCPU と GB のメモリが備えら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は、Docker ホストは 4 つの vCPU と 4GB のメモリを備えている必要があります。

Compose ファイルは、一般的に次のような形式に従います:

```
    <service-name>:
    container_name:
    image: <repository>:<tag>
    ports:
        - 22
        - 9080
        - 9443
        - 161/udp
        - 35021-35030
    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" }
 '
<!--NeedCopy-->
```

```
    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
<!--NeedCopy-->
```

NetScaler CPX インスタンスの NetScaler ADM への追加

March 25, 2024

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場合は、NetScaler アプリケーション配信管理（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 を初めて設定するときに追加することも、後で追加することもできます。

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て各インスタンスのホスト名または IP アドレス、または 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 ADM 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。インスタンスの種類ごとにデフォルトのプロファイルが用意されています。たとえば、`ns-root-profile` は NetScaler ADC インスタンスのデフォルトのプロファイルです。このプロファイルは、デフォルトの ADC 管理者資格情報によって定義されます。インスタンスのデフォルトの管理者資格情報を変更した場合は、それらのインスタンスのカスタムのインスタンスプロファイルを定義できます。インスタンスが検出された後にインスタンスの資格情報を変更した場合は、インスタンスプロファイルを編集、または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

以下の点について確認してください。

- NetScaler ADM ソフトウェアを Citrix XenServer にインストールしました。詳細については、NetScaler ADM ドキュメント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 ブラウザーで、**NetScaler** アプリケーション配信管理の **IP** アドレス（例:）を入力します。`http://192.168.100.1`
2. [**User Name**] と [**Password**] の各フィールドに管理者の資格情報を入力します。デフォルトの管理者の資格情報は **nsroot** と **nsroot** です。
3. [ネットワーク] **>** [インスタンス] **** [****NetScaler**] に移動し、[**CPX**] タブをクリックします。
4. [**Add**] をクリックして、NetScaler ADM に新しい CPX インスタンスを追加します。
5. [**NetScaler CPX** の追加] ページが開きます。次のパラメーターの値を入力します:

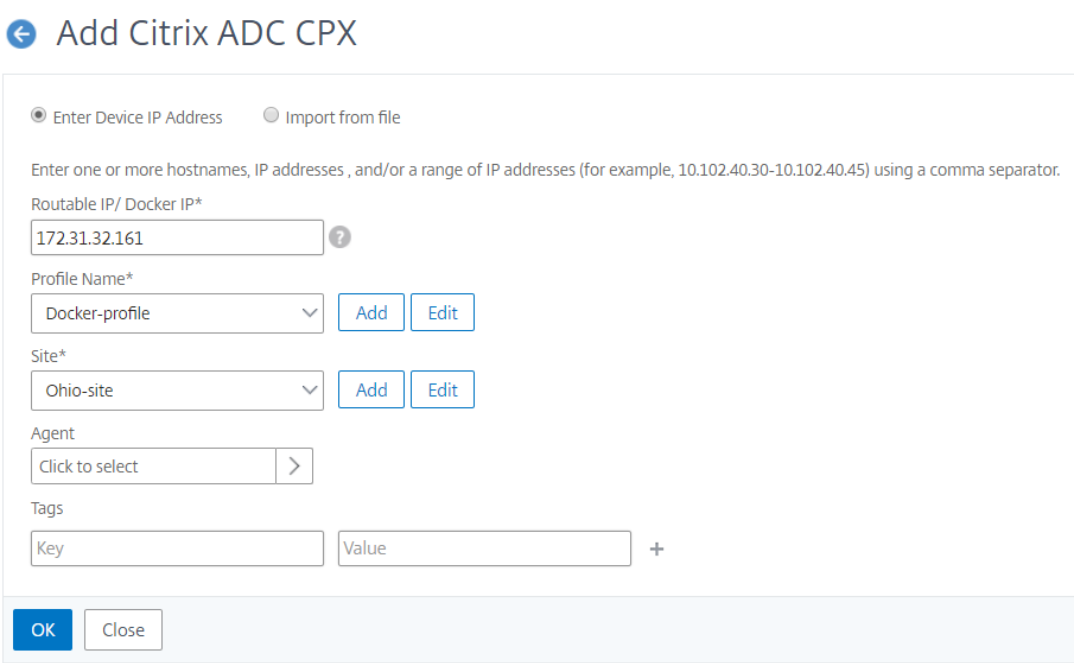
   a) CPX インスタンスの到達可能な IP アドレス、または CPX インスタンスがホストされている Docker コンテナの IP アドレスのいずれかを指定することにより、CPX インスタンスを追加できます。

   b) CPX インスタンスのプロファイルを選択します。

   c) インスタンスを展開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、キーと値のペアをインスタンスに入力できます。キーと値のペアを追加すると、後で簡単にインスタンスを検索できます。



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て、[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 ADM** に追加する

環境変数を使用して NetScaler CPX インスタンスを NetScaler ADM 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 - ノードの IP アドレス
- NS_HTTP_PORT - ノードにマップされた HTTP ポート
- NS_HTTPS_PORT - ノードにマップされた HTTPS ポート
- NS_SSH_PORT - ノードにマップされた SSH ポート
- NS_SNMP_PORT - ノードにマップされた SNMP ポート
- NS_ROUTABLE -（NetScaler CPX ポッドの IP アドレスは外部からルーティングできません。）
- NS_MGMT_USER –ADM ユーザー名
- NS_MGMT_PASS –ADM パスワード

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

```
1  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
2
3  <!--NeedCopy-->
```

Kubernetes ConfigMap を使用して **NetScaler CPX** インスタンスを **NetScaler ADM** に追加する方法

NetScaler CPX は、Kubernetes ConfigMaps を介してボリュームマウントされたファイルを使用して NetScaler ADM に登録すること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 でいくつかの環境変数を指定する必要があります。これらの変数は、ConfigMaps および Secrets を使用してボリュームマウントファイルとともに指定する必要があります。

必要な環境変数とその説明は次のとおりです:

- NS_HTTP_PORT: ノードにマップされた HTTP ポートを指定します。
- NS_HTTPS_PORT: ノードにマップされた HTTPS ポートを指定します。
- NS_SSH_PORT: ノードにマップされた SSH ポートを指定します。
- NS_SNMP_PORT: ノードにマップされた SNMP ポートを指定します。

NetScaler CPX には、記載されている環境変数とは別に、登録対象の ADM エージェントに関する情報が必要です。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 エージェントの認証情報を含む 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を使用して、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/adminfo/server/
            name: adm-agent-config
          - mountPath: /var/adminfo/credentials/
            name: adm-agent-user
          ...
    volumes:
    ...
    - name: adm-agent-config
      configMap:
        name: adm-agent-config
    - name: adm-agent-user
      secret:
        secretName: adm-secret
```

上の例では、adm-agent-configという名前の ConfigMap と Secretadm-agent-user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、次の形式の ADM エージェントの IP アドレスまたは FQDN が必要です。

```
NS_MGMT_SERVER=adm-agent
```

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

次のコマンドを使用して、ConfigMap を作成します:

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

注: ファイル名には`adm_reg_envs`変数が必要であり、次のパスにマウントする必要があります: `/var/adminfo/server/`。

Secret: 次のコマンドを使用して、Kubernetes の Secret を作成します。次のコマンドで、`user123`は ADM エージェントのユーザー名、`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、ADM エージェントをクラスターにデプロイする前でも、必要な環境変数とボリュームマウント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 エージェントを展開する前に NetScaler CPX インスタンスを展開すると、NetScaler CPX は ADM エージェントが展開されるまで登録を試み続けます。ADM エージェントが展開されると、NetScaler CPX インスタンスは環境変数とボリュームマウントファイルを介し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 の再展開を回避できます。

ADM エージェントにすでに登録されている 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、すでにデプロイされている NetScaler CPX の ConfigMap とシークレットの構成情報を更新できます。ConfigMap の作成元のファイルを新しい 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 Secret を削除し、新しい ADM エージェントの資格情報を使用して新しい Secret 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

November 23, 2023

現在、NetScaler CPX は NetScaler ADM サーバーからライセンスを取得しています。Kubernetes 環境では、NetScaler CPX は動的に稼働または停止することがあります。NetScaler CPX が予期せず停止した場合、NetScaler ADM サーバーがライセンスを解放するのに数分かかります。NetScaler ADM サーバーは、NetScaler CPX がダウンしたときにすぐにそのようなライセンスを再利用できるようにする必要があります。そうすれば、同じライセンスを別の NetScaler ADC CPX に割り当てることができます。また、何らかの理由で NetScaler ADM サーバーに到達できない場合、クラスター内で新しい NetScaler ADC CPX ライセンスを取得することができません。

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 クラスター内の 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 クラスターに導入された 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の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 CPX がダウンしても、NetScaler ADM サーバーはすぐにライセンスを取り戻すことができます。

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、NetScaler CPX のサイドカーとスタンドアロン展開の両方で利用できます。

> 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 の以前のバージョンのライセンスをサポートしていません。

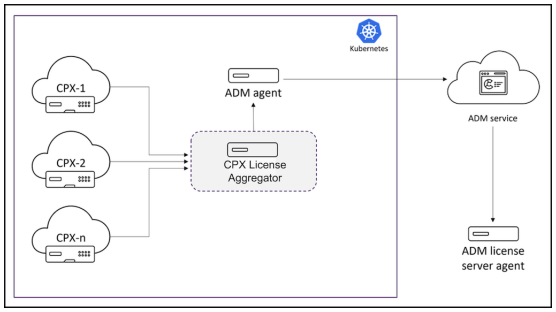
NetScaler CPX ライセンスアグリゲーターの主な利点

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 ADM ライセンスサーバーは、最大 10000 の NetScaler CPX デプロイ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 ADM ライセンスサーバーに対して単一のクライアントとして機能できます。したがって、単一の NetScaler ADM ライセンスサーバーで多数の NetScaler ADC CPX を拡張できます。
- 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 ADM 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 ADM 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、設定した待機期間の後にそのような NetScaler CPX からライセンスを取り戻すことができます。

デプロイトポロジー

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



この図の説明:

- CPX NetScaler CPX

- `CPX License Aggregator` NetScaler CPX ライセンスアグリゲーターを意味します

このサンプル展開では、NetScaler CPX ライセンスアグリゲーターサービスが NetScaler CPXS および NetScaler ADM 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 ライセンスアグリゲーターサービスは、NetScaler CPX と NetScaler ADM エージェント間の仲介役として機能し、クラスター内のすべての NetScaler CPX を監視し、それらのライセンスを管理します。

ヘルムチャートを使用して **NetScaler CPX** ライセンスアグリゲーターを導入

前提条件

次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 ライセンスアグリゲーターで Redis データベースのパスワードを入力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のパスワードを指定できます。次のコマンドを使用してシークレットを作成できます:

```
kubectl create secret generic dbsecret --from-literal=password=<
    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 ADC CPX ライセンスアグリゲーターを展開するには、次の手順を実行してください。さまざまなタイプの NetScaler ADC CPX ライセンスの詳細については、「NetScaler CPX ライセンス」を参照してください。

Helm チャートのインストール　次のコマンドを使用して、NetScaler CPX ライセンスアグリゲーターの 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 ADC CPX** ライセンスアグリゲーターのインストール　所有している NetScaler ADC 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:
>
> 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=<License-Server-IP-or-FQDN>,redis.
    secretName=<Kubernetes-Secret-for-DB-password>,licenseAggregator
    .username=<unique-ID-for-CLA>,licenseInfo.instanceQuantum=<
    QUANTUM>,licenseInfo.instanceLowWatermark=<LOW WATERMARK>,
    licenseInfo.bandwidthPlatinumQuantum=<QUANTUM-in-Mbps>,
    licenseInfo.bandwidthPlatinumLowWatermark=<LOW WATERMARK-in-Mbps
    >
```

エンタープライズ帯域幅エディションの場合:

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

標準帯域幅エディションの場合:

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

これらのコマンドは、NetScaler CPX ライセンスアグリゲーターをデフォルト構成で Kubernetes クラスターにデプロイします。インストール時にパラメーターを設定できます。詳しくは、Helm チャート GitHub リポジトリの **NetScaler CPX** ライセンスアグリゲーター構成セクションを参照してください。このセクションには、インストール中に構成できる必須およびオプションのパラメーターが一覧表示されています。

vCPU ライセンスを管理するための **NetScaler ADC CPX** ライセンスアグリゲーターのインストール

所有している NetScaler ADC 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:
デフォルトでは、Helm チャートは推奨される RBAC 役割と役割のバインド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=<License-Server-IP-or-FQDN>,redis.
   secretName=<Kubernetes-Secret-for-DB-password>,licenseAggregator
   .username=<unique-ID-for-CLA>,licenseInfo.vcpuPlatinumQuantum=<
   QUANTUM>,licenseInfo.vcpuPlatinumLowWatermark=<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=<License-Server-IP-or-FQDN>,redis.
   secretName=<Kubernetes-Secret-for-DB-password>,licenseAggregator
   .username=<unique-ID-for-CLA>,licenseInfo.vcpuEnterpriseQuantum
   =<QUANTUM>,licenseInfo.vcpuEnterpriseLowWatermark=<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=<License-Server-IP-or-FQDN>,redis.
   secretName=<Kubernetes-Secret-for-DB-password>,licenseAggregator
   .username=<unique-ID-for-CLA>,licenseInfo.vcpuStandardQuantum=<
   QUANTUM>,licenseInfo.vcpuStandardLowWatermark=<LOW WATERMARK>
```

複数のライセンスを管理するための **NetScaler ADC CPX** ライセンスアグリゲーターのインストール

複数の種類のライセンスを管理するために NetScaler ADC CPX ライセンスアグリゲーターが必要な場合は、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:

NetScaler CPX ライセンスアグリゲーターを`pooled platinum bandwidth edition`および`vCPU platinum edition`ライセンス用に展開するには:

```
helm install demo citrix/cpx-license-aggregator --set
   licenseServer.address=<License-Server-IP-or-FQDN>,redis.
   secretName=<Kubernetes-Secret-for-DB-password>,
   licenseAggregator.username=<unique-ID-for-CLA>,licenseInfo.
   instanceQuantum=<QUANTUM>,licenseInfo.instanceLowWatermark=<
   LOW WATERMARK>,licenseInfo.bandwidthPlatinumQuantum=<QUANTUM-
   in-Mbps>,licenseInfo.bandwidthPlatinumLowWatermark=<LOW
   WATERMARK-in-Mbps>,licenseInfo.vcpuPlatinumQuantum=<QUANTUM>,
   licenseInfo.vcpuPlatinumLowWatermark=LOW WATERMARK>
```

NetScaler CPX ライセンスアグリゲーターからライセンスを取得するように **NetScaler ADC CPX** を構成

NetScaler CPX ライセンスアグリゲーターを使用して NetScaler CPX のライセンス供与を行う場合は、NetScaler CLA CPX デプロイメント YAML で環境変数を指定する必要があります。

NetScaler `ipaddressdomainname` CPX ライセンスアグリゲーターにアクセスできるまたはは、次のように、この環境変数で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 では、次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 単位で指定します。
- `Edition`: ライセンスのエディションを指定します。サポートされている値には、Standard、Platinum、および Enterprise が含まれます。
- `CPX_CORES` : NetScaler CPX で実行するコアの数を指定します。

さまざまな NetScaler ADC CPX ライセンスオプションについて詳しくは、「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
```

また、NetScaler CPX YAML に次のラベルを追加する必要があります。

```
      labels:
        adc: citrix
```


NetScaler CPX ライセンスアグリゲーターのサンプル展開については、「NetScaler CPX ライセンスアグリゲーター: 展開例」を参照してください。

NetScaler CPX の構成

March 25, 2024

NetScaler CPX インスタンスを構成するには、Linux Docker ホストから CLI プロンプトにアクセスするか、NetScaler NITRO API を使用します。

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

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 インスタンスへのログオンに使用する、デフォルトの管理者資格情報は root/linux です。

```
root@ubuntu:~# ssh -p 32777 root@127.0.0.1
root@127.0.0.1's password:
Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.19.0-25-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Tue Dec 15 02:45:42 2015 from 172.17.0.1
root@10:~#
```

次のコマンドを入力して、インスタンスのコマンドラインプロンプトで CLI コマンドを実行します。**cli_script.sh** “<command>”

例:

```
root@10:~# cli_script.sh "show ip"
exec: show ip
        Ipaddress        Traffic Domain  Type
        ---------        --------------  ----
1)      172.17.0.4       0               NetScaler IP|VIP
2)      192.0.0.1        0               SNIP
```

インスタンスの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 でのデフォルト以外のパスワードの使用のサポート

NetScaler CPX では、root アカウントにデフォルト以外のパスワードを使用することをサポートしています。`nsroot`。NetScaler CPX が展開されると、デフォルトのパスワードが生成され、ユーザーに割り当てられます。このデフォルトのパスワードは、SSH ユーザー用（`root`および`nsroot`）でも更新されます。このデフォルトの

パスワードは手動で変更できます。rootおよびnsrootユーザーアカウントのデフォルトの SSH パスワードを手動でリセットすることもできます。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お勧めします。

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

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

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

たとえば、IP アドレスとユーザーnsrootおよびパスワードCitrix123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:

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

NITRO API を使用して NetScaler CPX インスタンスを構成する

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** ブラウザー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** ブラウザーで次のように入力します。:

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

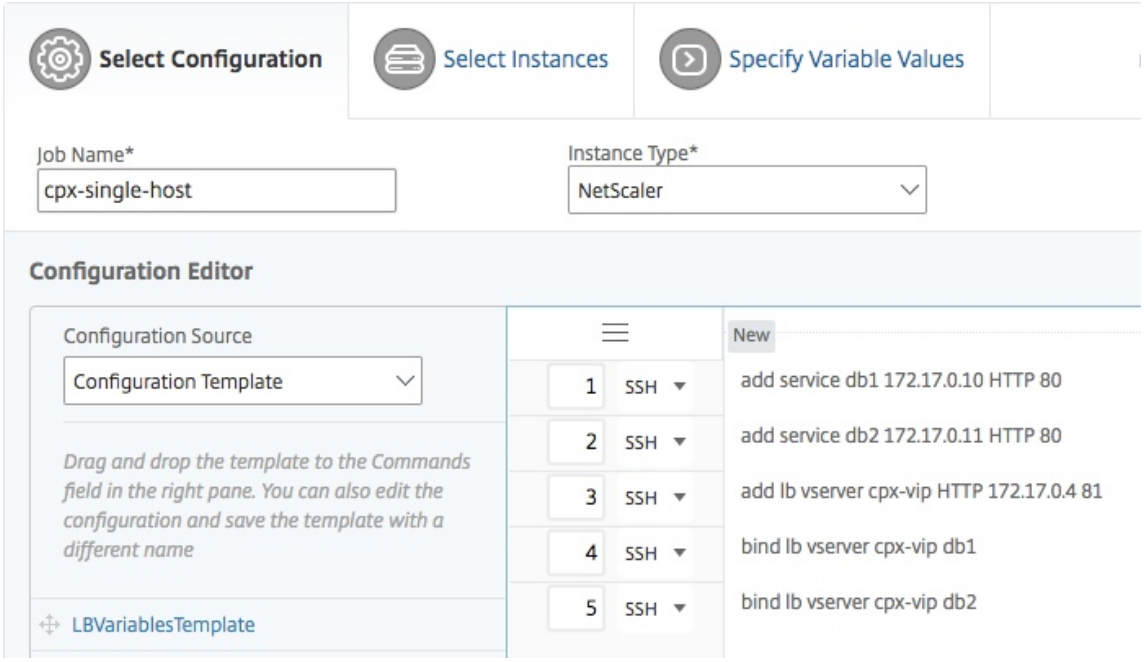
NITRO API の使用について詳しくは、「REST Web サービス(REST Web Services)」を参照してください。NetScaler CPX の場合は、netscaler-ip-addressが記載されている場所でCPX IP address:portを使用してください。

ジョブを使用した NetScaler CPX インスタンスの構成

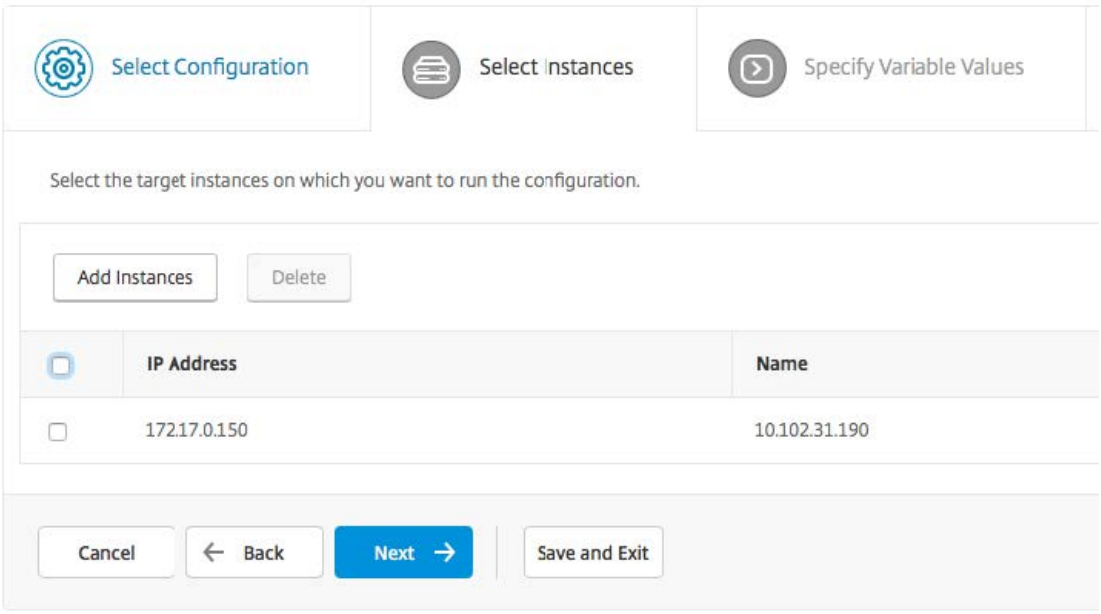
NetScaler CPX インスタンスを構成するには、NetScaler ADM でジョブを作成して実行します。構成テンプレートの構成を使用する、ほかのデバイスで利用可能な構成を抽出する、テキストファイルに保存された構成を使用するなどの方法があります。また、別のインスタンスで構成ユーティリティを使用して行われた構成を記録することもできます。その後、NetScaler ADM は、NetScaler CPX インスタンスで使用できる対応する CLI コマンドを表示します。構成を選択したら、構成をロードする **NetScaler CPX** インスタンスを選択し、変数値を指定してジョブを実行する必要があります。

ジョブを使用して **NetScaler CPX** インスタンスを構成するには:

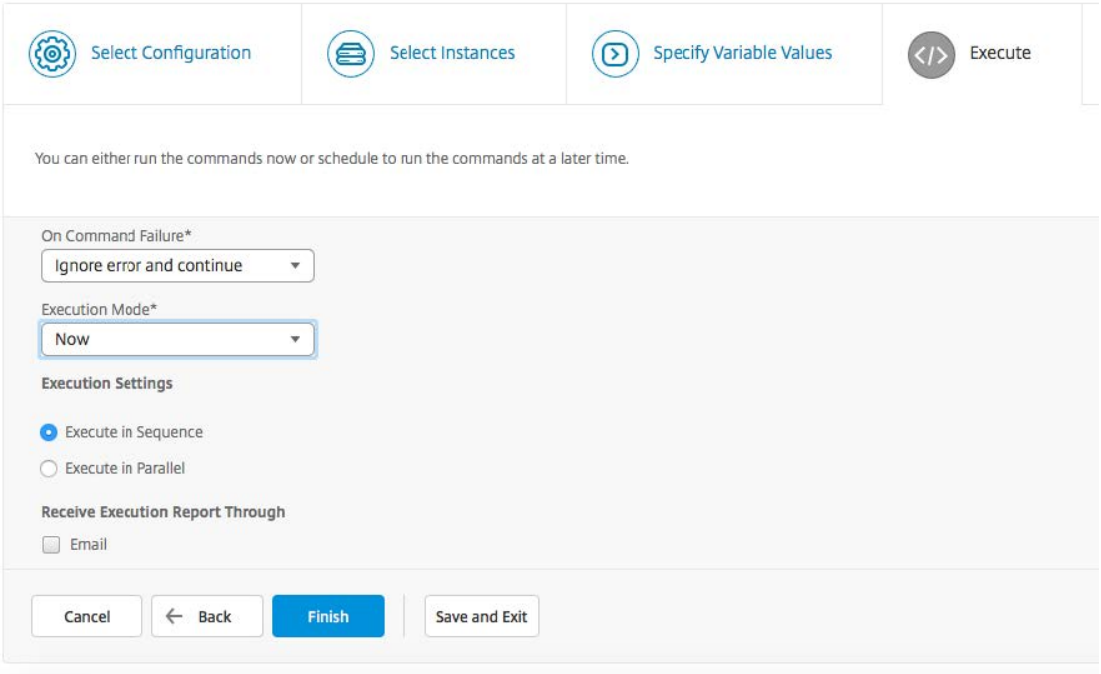
1. 管理者の認証情報を使用して NetScaler ADM にログオンします。
2. [**Networks**] > [**Configuration Jobs**] の順に選択してから、[**Create Job**] をクリックします。
3. 必要な値を指定して、構成ソースを選択します。実行するコマンドを入力することもできます。



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



5. 実行設定を指定して [Finish] をクリックすると、NetScaler CPX インスタンスでコマンドが実行されます。設定を保存して後で実行する場合は、[保存して終了] をクリックします。



NetScaler CPX インスタンスでの AppFlow の構成

November 23, 2023

NetScaler CPX インスタンスで AppFlow 機能を構成して、Web ページのパフォーマンスデータ、フローとユーザーセッションレベルの情報、およびアプリケーションパフォーマンスの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 ADM に送信され、そこですべてのアプリケーションのリアルタイムレポートと履歴レポートを表示できます。

AppFlow を構成するには、最初に 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のコレクターを指定します。その後、構成済み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、アクションを各ポリシーに関連付けます。このポリシーは、NetScaler CPX に対し、フローレコードが関連するアクションに送信されるリクエストを選択するように指示します。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、ポリシーを有効にします。

さらに、AppFlow パラメーターを設定して、テンプレート更新間隔を指定し、`httpURL`、`httpCookie`、および`httpReferer`情報のエクスポートを有効にすることができます。各コレクターで、NetScaler CPX IP アドレスをエクスポーター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一連のコレクター（アクション）にエクスポートする方法が正確に決まります。コマンドラインインターフェースには、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が用意されています。

レコードを監視する前に、NetScaler CPX インスタンスを NetScaler ADM に追加する必要があります。NetScaler CPX インスタンスを NetScaler ADM に追加する方法の詳細については、「NetScaler ADM を使用して NetScaler CPX インスタンスをインストールする」を参照してください。

AppFlow を有効にする

AppFlow 機能を使用するには、最初に機能を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 メッセージをリスンします。コレクターを構成するときに、デフォルトの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:

次のコマンドを使用して、コレクターを追加します:

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

構成を確認するには、次のコマンドを使用します:

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

コマンドラインインターフェイスを使用して複数のコレクターを指定するには:

次のコマンドを使用して、同じデータを追加し、複数のコレクターに送信します:

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

add appflow collector <collector2> -IPAddress <IP> -Transport Logstream

add appflow action <action> -collectors <collector1> <collector2> -
   Transport Logstream

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

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


AppFlow アクションの構成

AppFlow アクションはセットコレクターであり、関連付けられた AppFlow ポリシーが一致した場合のフローレコードの送信先です。

次のコマンドを使用して、AppFlow アクションを構成します:

```
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後、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規則に基づいています。

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、AppFlow ポリシーを作成して構成を確認します:

```
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、ポリシーを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:

次のコマンドを使用して、AppFlow ポリシーをグローバルにバインドします:

```
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た **NetScaler CPX** の構成

November 23, 2023

コマンドラインインターフェイス（`cli_script.sh`）、NITRO API、または NetScaler ADM 構成ジョブを使用して NetScaler CPX を構成する代わりに、NetScaler CPX インスタンスを展開するとき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 コンテナを展開する時に、静的な構成ファイルを入力ファイルとして提供できます。NetScaler CPX コンテナの起動時に、コンテナは静的構成ファイルに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 CPX コンテナで動的に実行できる NetScaler 固有の構成および bash シェルコマンドが含まれています。

静的構成ファイルの構造

前述のように、NetScaler CPX を導入すると、静的構成ファイルに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`タグの下で、NetScaler CPX 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、NetScaler CPX で実行する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、構成ファイルに指定された順序でコンテナ上で実行されます。

> 重要:
>
> - このタグは、構成ファイル内で複数回繰り返すことができます。
> - タグで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している必要があります。
> - 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「#」の文字を追加してください。
> - NetScaler CPX コンテナを構成ファイルとともに展開する時に障害が発生すると、それらの障害はコンテナ内の`ns.log`ファイルに記録されます。
> - NetScaler CPX コンテナを再起動すると、構成ファイルがコンテナに再適用されます。

```
#NetScaler Commands

add lb vserver v1 http 1.1.1.1 80

add service s1 2.2.2.2 http 80

bind lb vserver v1 s1

```



```
#Shell Commands

touch /etc/a.txt

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

#NetScaler Commands

add lb vserver v2 http

#Shell Commands

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

#NetScaler Commands

add lb vserver v3 http

#This is a test configuration file
<!--NeedCopy-->
```

NetScaler CPX コンテナをインストールし、NetScaler CPX コンテナを構成ファイルに基づいて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
<!--NeedCopy-->
```

NetScaler CPX でのダイナミックルーティングのサポート

November 23, 2023

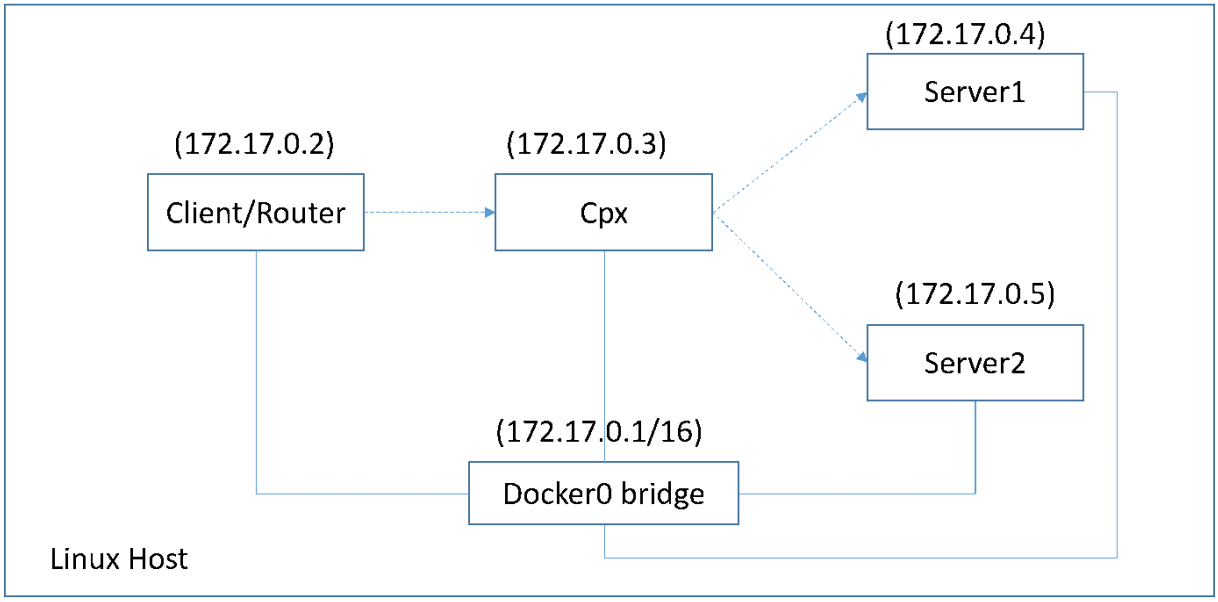
NetScaler CPX は BGP 動的ルーティングプロトコルをサポートしています。動的ルーティングプロトコルの主な目的は、仮想サーバーにバインドされたサービスの状態に基づいて、仮想サーバーの IP アドレスをアドバタイズすることです。これは、アップストリームルーターが、地形的に分散した仮想サーバーへの複数のルートから最適なルートを選択するのに役立ちます。

NetScaler CPX のデフォルト以外のパスワードについては、「NetScaler CPX の構成」`Support for using a non-default password in NetScaler CPX` ドキュメントの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 インスタンスが、同じ Docker ホストのコンテナとして展開されます。コンテナはすべて docker0 ブリッジで接続されます。この環境で、NetScaler CPX インスタンスは、同一の Docker ホストでコンテナとしてプロビジョニングされているアプリケーションのプロキシとして機能します。NetScaler CPX ホストネットワークモードの展開について詳しくは、「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、シングルホストトポロジを示しています。



このトポロジでは、仮想サーバーが構成され、BGP を使用してアップストリームネットワークまたはルーターに（サービスの状態に基づいて）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 で 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ーを作成します。

```
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. 次のコマンドを実行してコンテナにログインします:

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

3. 次のコマンドを実行して、BGP 機能を有効にします:

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

4. `show ns ip`コマンドを実行して NSIP を取得します。


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

5. 次のコマンドを実行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、サービスを仮想サーバーにバインドします。

7. 次のコマンドを実行して、VIP のhostrouteを有効にします:

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

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

8. BGP ルーターを構成します:

たとえば、次のように構成する場合:

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

コマンドを次のように指定します:

```
 curl -u username:password http://<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 コマンドを実行して、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://<NSIP>:9080/nitro/v1/config/ --
   data 'object={
 "params":{
 "action":"apply" }
 ,"routerDynamicRouting": {
 "commandstring" : "ns route-install bgp" }
 }
 '
```

10. 次の NITRO コマンドを実行して、BGP 隣接状態を確認します:


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

出力例:

```
root@ubuntu:~# curl -u username:password http://172.17.0.3:9080/
   nitro/v1/config/routerDynamicRouting/bgpRouter
{
 "errorcode": 0, "message": "Done", "severity": "NONE", "
    routerDynamicRouting":{
"bgpRouter":[{
 "localAS": 100, "routerId": "172.17.0.3", "afParams": [ {
 "addressFamily": "ipv4"  }
, {
 "addressFamily": "ipv6"  }
 ], "neighbor": [ {
 "address": "172.17.0.2", "remoteAS": 101, "ASOriginationInterval
    ": 15, "advertisementInterval": 30, "holdTimer": 90, "
    keepaliveTimer": 30, "state": "Connect", "singlehopBfd":
    false, "multihopBfd": false, "afParams": [ {
 "addressFamily": "ipv4", "activate": true  }
, {
 "addressFamily": "ipv6", "activate": false  }
 ]
```

11. 次のコマンドを実行して、BGP を介して学習し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. 次のコマンドを実行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。

NetScaler CPX の高可用性の設定

November 23, 2023

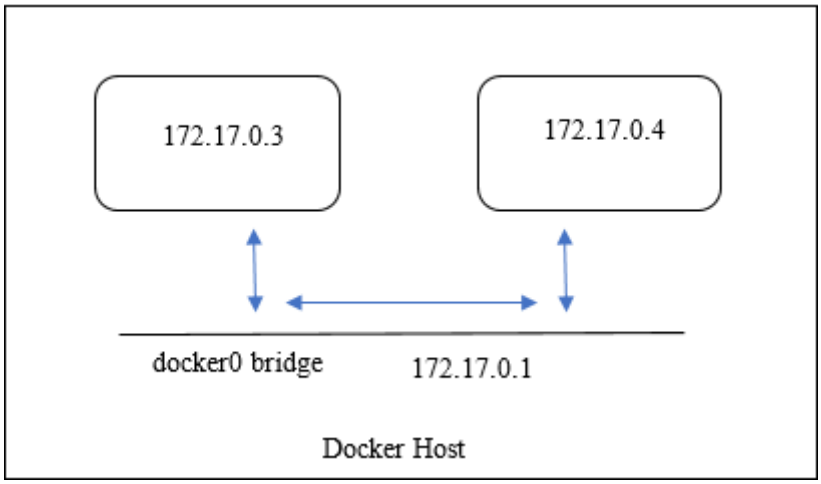
ミッションクリティカルおよびビジネスクリティカルなアプリケーションがインストールされたシステムは、単一障害点が発生することなく継続的に利用可能である必要があります。高可用性を備えたシステムは、ユーザーに提供するサービスを中断することなく、アプリケーションの継続的な可用性を保証します。NetScaler CPX は、2 つの NetScaler インスタンスの高可用性導入をサポートします。これにより、予期しないダウンタイムからサービスを保護し、障害発生時の事業継続性を確保できます。高可用性を構成したら、ユーザーへのサービスを中断することなく NetScaler CPX ソフトウェアをアップグレードすることもできます。

> 注:
> 内部ユーザーアカウントが無効になっている場合、NetScaler CPX の高可用性機能はサポートされません。

トポロジー **1: NetScaler CPX** インスタンスをブリッジネットワークモードの単一 **Docker** ホストにデプロイします

このトポロジーでは、ブリッジネットワークモードの同じ Docker ホストに 2 つの NetScaler CPX ノードが作成されます。両方のノードは同じブリッジネットワーク上にあり、ノードは互いに直接到達可能です。

次の図は、このトポロジを説明しています。



この例では、CPX-1（NSIP: 172.17.0.3）と CPX-2（NSIP: 172.17.0.4）の 2 つの NetScaler CPX インスタンスが同じ Docker ホスト上に作成されます。高可用性をサポートするには、もう一方のノードの NSIP を使用して、両方の NetScaler CPX インスタンスで高可用性ノードを構成する必要があります。

ブリッジモードの単一 Docker ホスト上の NetScaler CPX インスタンスで高可用性サポートを構成するには、次の手順を実行します。

1. Docker ホストにアクセスし、NetScaler CPX インスタンスの SSH プロンプトにログオンします。詳しくは、「コマンドラインインターフェイスを使用した NetScaler CPX インスタンスの構成」を参照してください。
2. 次のコマンドを使用して、CPX-1 インスタンスに高可用性ノードを構成します。

```
cli_script.sh  'add ha node 1 172.17.0.4 [-inc enabled]'
```

3. 次のコマンドを使用して、CPX-2 インスタンスに高可用性ノードを構成します。

```
cli_script.sh  'add ha node 1 172.17.0.3 [-inc enabled]'
```

> 注:
> ブリッジネットワークモードの NetScaler CPX ノードを再起動すると、NetScaler CPX に割り当てられた IP アドレスは、ホスト上の Docker バージョンによって変わることがあります。NetScaler CPX の再起動後にい

> ずれかのノードの NSIP が変更された場合、構成を保存しても既存の高可用性構成は機能しません。その場合は、NetScaler CPX ノードで高可用性を再度構成する必要があります。

トポロジー **2:** ブリッジネットワークモードを使用して **NetScaler CPX** を異なる **Docker** ホストにデプロイします

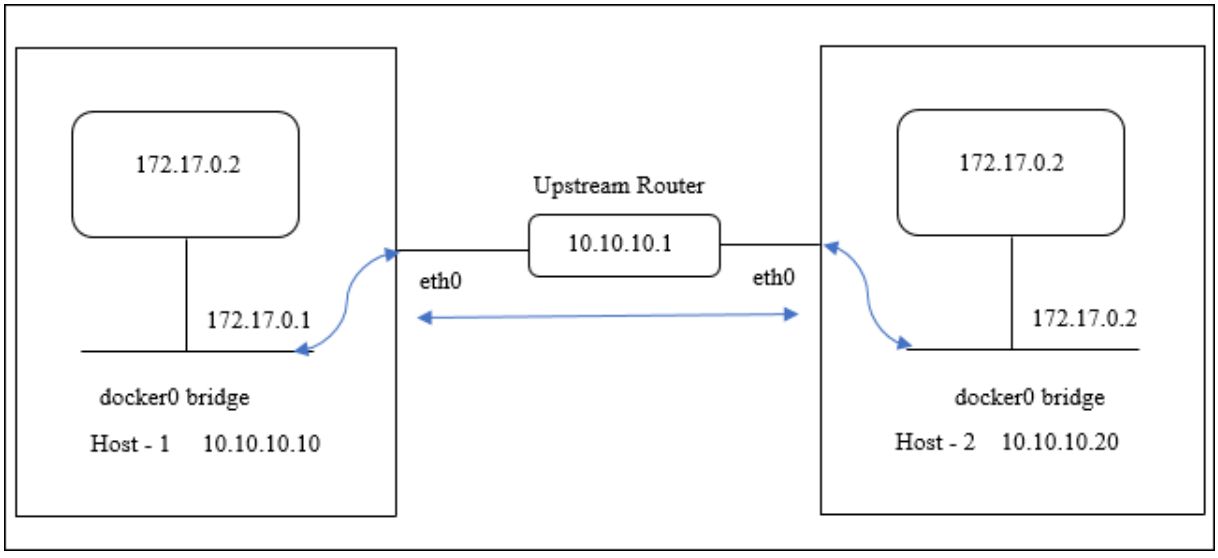
このトポロジーでは、2 つの NetScaler CPX インスタンスが、互いにアクセス可能な 2 つの異なる Docker ホストにブリッジモードでデプロイされます。この展開では、NetScaler CPX はホストの IP アドレスを認識している必要があります。HOST 環境変数は、NetScaler CPX をプロビジョニングするときに NetScaler **CPX** にホストの **IP** アドレスを認識させるために使用できます。

NetScaler CPX ノードのポートマッピングを設定する必要があります。NetScaler **CPX** ノードの作成時に **docker run** コマンドの**-p** オプションを使用して、必要なポートのポートマッピングを有効にできます。

次のポートをマッピングする必要があります:

- UDP 3003
- TCP 3008
- TCP 8873

次の図は、2 つの NetScaler CPX インスタンスを 2 つの異なる Docker ホストにブリッジモードで展開するトポロジを示しています。



この図では、青い直線は 2 つのホスト間の CPX-HA トラフィックのフローを表しています。

> 注: Docker ホストでは、高可用性ペアを形成できる NetScaler CPX は 1 つだけです。同じホスト上の他の NetScaler CPX は、別のホスト上の別の NetScaler CPX と高可用性ペアを形成することはできません。

NetScaler インスタンスを異なる Docker ホストにブリッジモードでデプロイし、サンプルトポロジーを使用して高可用性サポートを構成するには、次の手順を実行します。

この例では、HOST1 の IP アドレスは`10.10.10.10/24`として構成され、HOST2 の IP アドレスは`10.10.10.20/24`として構成されています。

1. 次のコマンドを使用して、必要なポートマッピングを含む NetScaler CPX をホスト 1 にデプロイします。

```
Docker run -dt --privileged=true -e EULA=yes --ulimit core=-1 -p
    8873:8873 -p 3003:3003/udp -p 3008:3008 -e Host=10.10.10.10 cpx
    :latest
```

2. ホスト 2 の IP アドレスで同じコマンドを使用して、ホスト 2 に NetScaler CPX をデプロイします。

```
docker run -dt --privileged=true -e EULA=yes --ulimit core=-1 -p
    8873:8873 -p 3003:3003/udp -p 3008:3008 -e HOST=10.10.10.20 cpx
    :latest
```

3. 次のコマンドを使用して、CPX-1 インスタンスに高可用性ノードを構成します。

```
cli_script.sh   'add ha node 1 10.10.10.20 -inc enabled'
```

4. 次のコマンドを使用して、CPX-2 インスタンスに高可用性ノードを構成します。

```
cli_script.sh  'add ha node 1 10.10.10.10 -inc enabled'
```

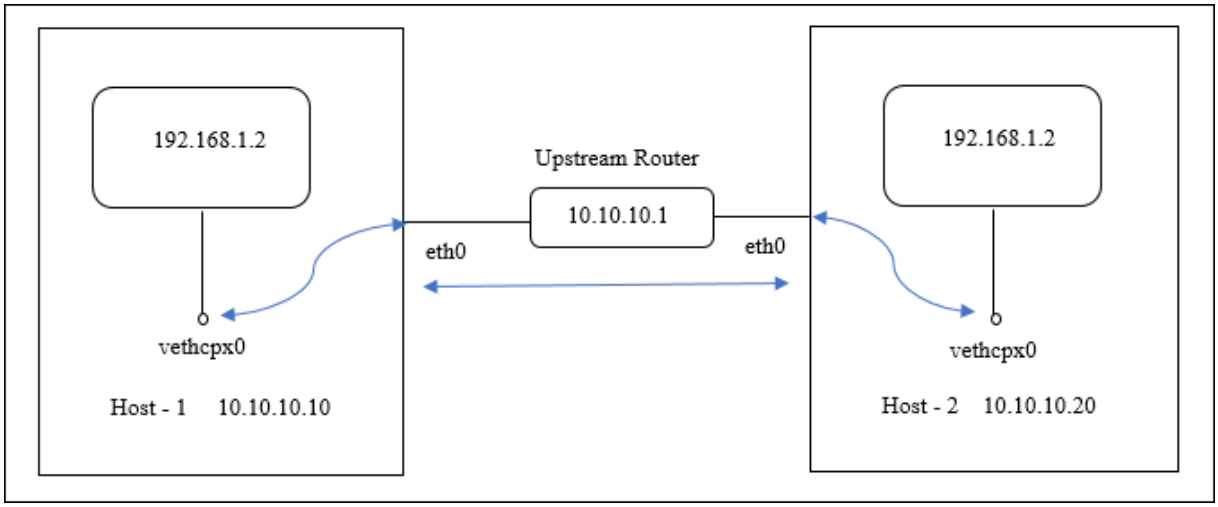
> 注: この展開では、高可用性ノードの NSIP アドレスではなく、高可用性ノードのホスト IP アドレスを使用する必要があります。

トポロジー **3:** 専用インターフェースを使用せずに、ホストネットワークモードで異なる **Docker** ホストに **NetScaler CPX** を導入

このトポロジーでは、2 つの NetScaler CPX インスタンスが、専用インターフェースのないホストモードで 2 つの異なる Docker ホストにデプロイされます。ホストは相互に到達可能である必要があります。
この展開では、NetScaler CPX はホストの IP アドレスを認識している必要があります。NetScaler CPX のプロビジョニング中に HOST 環境変数を使用して、ホストの **IP** アドレスを認識させることができます。

NetScaler CPX ノードのポートマッピングを設定する必要があります。NetScaler **CPX** ノードの作成時に **docker run** コマンドの**-p** オプションを使用して、必要なポートのポートマッピングを有効にできます。

次の図は、このトポロジについて説明しています。



この図では、青い直線は 2 つのホスト間の CPX-HA トラフィックのフローを表しています。

> 注: Docker ホストでは、ホストモードの NetScaler CPX を 1 つしかデプロイできません。

次の手順を実行して、NetScaler CPX インスタンスを展開し、サンプルトポロジーを使用して高可用性サポートを構成します。

1. 次のコマンドを使用して、NetScaler CPX を必要なポートマッピングでホスト 1 に展開します。

```
docker run -dt --privileged=true -e EULA=yes --ulimit core=-1 --
    net=host -e NS_NETMODE=HOST -e HOST=10.10.10.10 cpx:latest
```

2. 次のコマンドを使用して、ホスト 2 の IP アドレスを使用して NetScaler CPX をホスト 2 にデプロイします。

```
docker run -dt --privileged=true -e EULA=yes --ulimit core=-1
--net=host -e NS_NETMODE=HOST -e HOST=10.10.10.20 cpx:latest
```

3. 次のコマンドを使用して、CPX-1 インスタンスに高可用性ノードを構成します。

```
cli_script.sh   'add ha node 1 10.10.10.20 -inc enabled
```

4. 次のコマンドを使用して、CPX-2 インスタンスに高可用性ノードを構成します。

```
cli_script.sh  'add ha node 1 10.10.10.10 -inc enabled'
```

トポロジ **4**: ホストネットワークモードと専用のインターフェイスを使用して **CPX** を異なる **Docker** ホストに展開

このトポロジーでは、2 つの NetScaler CPX インスタンスがホストネットワークモードで異なる Docker ホストにデプロイされます。ホストには複数のインターフェイスが必要です。**CPX_NW_DEV** 環境変数を使用して **NetScaler CPX** の専用インターフェイスを指定できます。

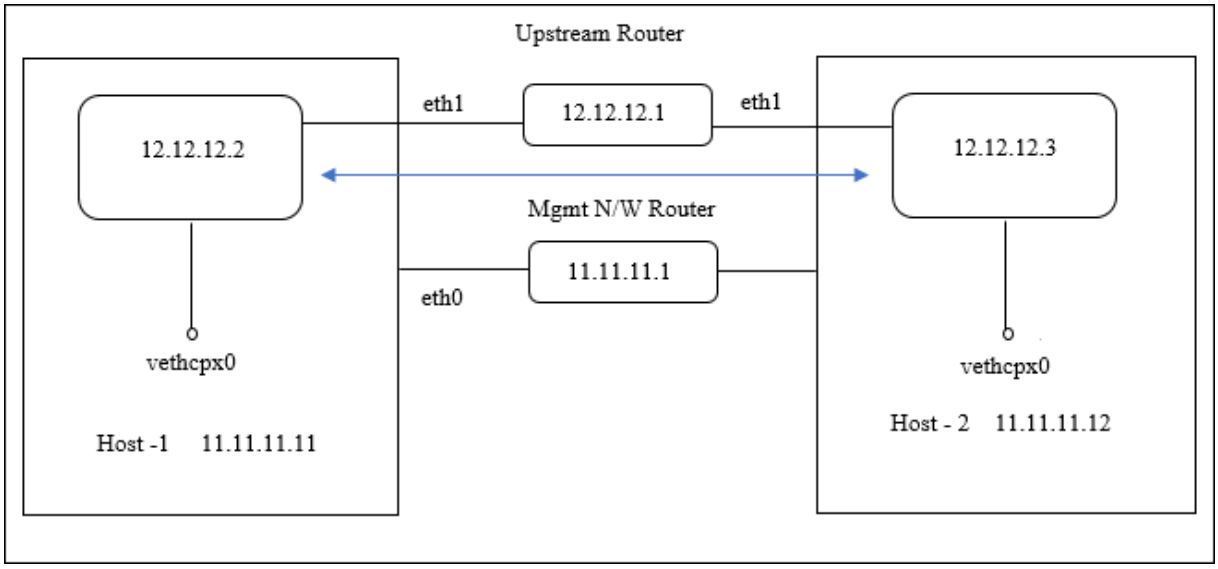
CPX_NW_DEV 環境変数を使用して NetScaler CPX に専用ネットワークインターフェイスを割り当てる方法について詳しくは、「docker run コマンドを使用した NetScaler CPX インスタンスのデプロイ」を参照してください。

異なる Docker ホストにデプロイされた NetScaler CPX は、専用インターフェースを使用してこのデータネットワーク上で相互にアクセスできる必要があります。

この構成により、高可用性ノードは、ポート 3003、3008、および 8873 で直接通信することにより、ハートビートメッセージを交換し、構成ファイルを同期できます。ホストに NAT 規則は必要ありません。ホストモードで作成された NetScaler CPX のデフォルトの NSIP は、両方のノードで同じです。そのため、**NS_IP** と **NS_GATEWAY** 情報も指定する必要があります。

この例では、2 つのホストモードの NetScaler CPX が 2 つの異なるホスト上に作成されます。NetScaler **CPX** インスタンスは両方のホストの **eth1** インターフェイスを所有し、**eth1** インターフェイスは同じネットワークに接続されます。

次の図は、このトポロジについて説明しています。この図の青い矢印は、eth1 インターフェイスに接続されたネットワーク上の CPX-HA トラフィックのフローを表しています。



> 注: Docker ホストでは、ホストモードの NetScaler CPX を 1 つしかデプロイできません。

次の手順を実行して、NetScaler CPX インスタンスを展開し、サンプルトポロジーを使用して高可用性サポートを構成します。

1. 次のコマンドを使用して、NetScaler CPX をホスト 1 でホストモードで展開します。

```
docker run -dt --privileged=true --net=host -e NS_NETMODE="HOST" -
    e CPX_NW_DEV=eth1 -e NS_IP='12.12.12.2' -e NS_GATEWAY='
    12.12.12.9' -e EULA=yes --ulimit core=-1 cpx:latest
```

2. 次のコマンドを使用して、NetScaler CPX をホスト 2 にホストモードでデプロイします。


```
docker run -dt --privileged=true --net=host -e NS_NETMODE="HOST" -
    e CPX_NW_DEV=eth1 -e NS_IP='12.12.12.3' -e NS_GATEWAY='
    12.12.12.10' -e EULA=yes --ulimit core=-1 cpx:latest
```

> 注：ハートビートメッセージを交換したり、構成ファイルを同期したりするには、両方の NetScaler CPX ノードがもう一方の NetScaler CPX ノードに到達するように静的ルートを構成する必要があります。

3. 次のコマンドを使用して、CPX-1 インスタンスに高可用性ノードを構成します。

```
cli_script.sh   'add ha node 1 12.12.12.3 [-inc enabled]'
```

4. 次のコマンドを使用して、CPX-2 インスタンスに高可用性ノードを構成します。

```
cli_script.sh  'add high availability node 1 12.12.12.2 [-inc
    enabled]'
```

Docker ログドライバーの構成

November 23, 2023

Docker は、実行中のコンテナから情報を取得するための「ログドライバー」というログメカニズムを備えています。生成されたログを Docker ログドライバーに転送するように NetScaler CPX コンテナを構成できます。Docker ロギングドライバーについて詳しくは、Configure logging drivers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るすべてのログが 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、--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 ログドライバーに転送するように構成できます。ログドライバーに構成可能なパラメーターがある場合は、--log-opt <NAME>=<VALUE>オプションを使用して設定できます。

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

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

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

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



```
<!--NeedCopy-->
```

NetScaler CPX インスタンスのアップグレード

November 23, 2023

NetScaler CPX インスタンスをアップグレードするには、NetScaler CPX インスタンスをシャットダウンし、同じマウントポイントに最新バージョンをインストールして、旧バージョンのインスタンスを削除します。マウントポイントは、ホスト上の **/cpx** ディレクトリのマウント先となるディレクトリです。

たとえば、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した場合、マウントポイントは **/var/cpx** になり、NetScaler CPX のマウントディレクトリは以下のように **/cpx** になります:

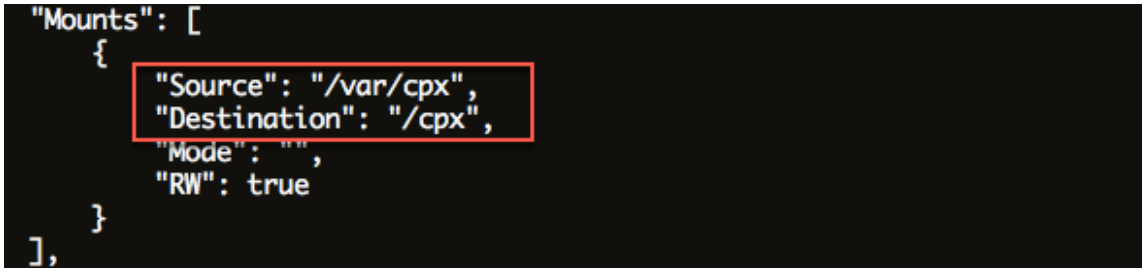
```
    root@ubuntu:~# docker run -dt -e EULA=yes --name mycpx -v /var/cpx
       :/cpx --ulimit core=-1 cpx:13.0-x.x
<!--NeedCopy-->
```

前提条件

以下を用意してください:

- 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。docker inspect <containerName>コマンドを使用して、ホストディレクトリの詳細情報を表示できます。<containerName>は NetScaler CPX コンテナ名です。

  このコマンドにより、ボリュームなどのコンテナの構成に関する詳細情報が出力されます。「**Mounts**」エントリの「**Source**」サブエントリに、そのホスト上のホストディレクトリの場所が表示されます。



- 最新の NetScaler CPX Docker イメージファイルをダウンロードし、NetScaler CPX Docker イメージをロードします。イメージをロードするには、Docker イメージファイルを保存したディレクトリに移動します。docker load -i <image_name>コマンドを使用してイメージをロードします。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
<!--NeedCopy-->
```

NetScaler CPX インスタンスをアップグレードするには

1. docker stop <containerName>コマンドで既存の NetScaler CPX インスタンスを停止します。<containerName>は NetScaler CPX インスタンス名です。

   ```
   root@ubuntu:~# docker stop mycpx
   mycpx
   <!--NeedCopy-->
   ```

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。インスタンスは必ず既存の NetScaler CPX インスタンスで使用していたものと同じマウントポイント（例: /var/cpx:/cpx）に展開してください。

   ```
   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
   <!--NeedCopy-->
   ```

   docker psコマンドを使用すると、展開した NetScaler CPX インスタンスが最新バージョンかどうかを確認できます。

   ```
   ```
       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
   <!--NeedCopy--> ```
   ```

3. 正しい NetScaler CPX インスタンスを展開できたことを確認したら、**docker rm** \< コンテナ名\> コマンドで旧バージョンのインスタンスを削除します。

   ```
   root@ubuntu:~# docker rm mycpx
   mycpx
   <!--NeedCopy-->
   ```


   ```

# 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

November 23, 2023

NetScaler インスタンスのプロビジョニング時に、Docker エンジンによって、1 つのプライベート IP アドレス（単一 IP アドレス）のみが NetScaler CPX インスタンスに割り当てられます。NetScaler インスタンスの 3 つの IP 機能が 1 つの IP アドレス上で多重化されます。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。

Docker エンジンによって割り当てられる単一の IP アドレスは動的です。この単一 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）仮想サーバーまたはコンテンツスイッチ（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る仮想サーバーは、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の作成時に、NetScaler CPX が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、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 環境では、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 で仮想サーバーを構成する必要がなくなります。

注意事項:

- 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で使用されていないことを確認します。
- ワイルドカード仮想サーバーに割り当てたポート番号が内部サービスで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では *（アスタリスク）文字はサポートされません。

負荷分散のワイルドカード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:

```
1   add lb vserver <name> <serviceType> 127.0.0.1 <port>
2
3   add lb vserver testlbvserver HTTP 127.0.0.1 30000
4 <!--NeedCopy-->
```

コンテンツスイッチのワイルドカード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。

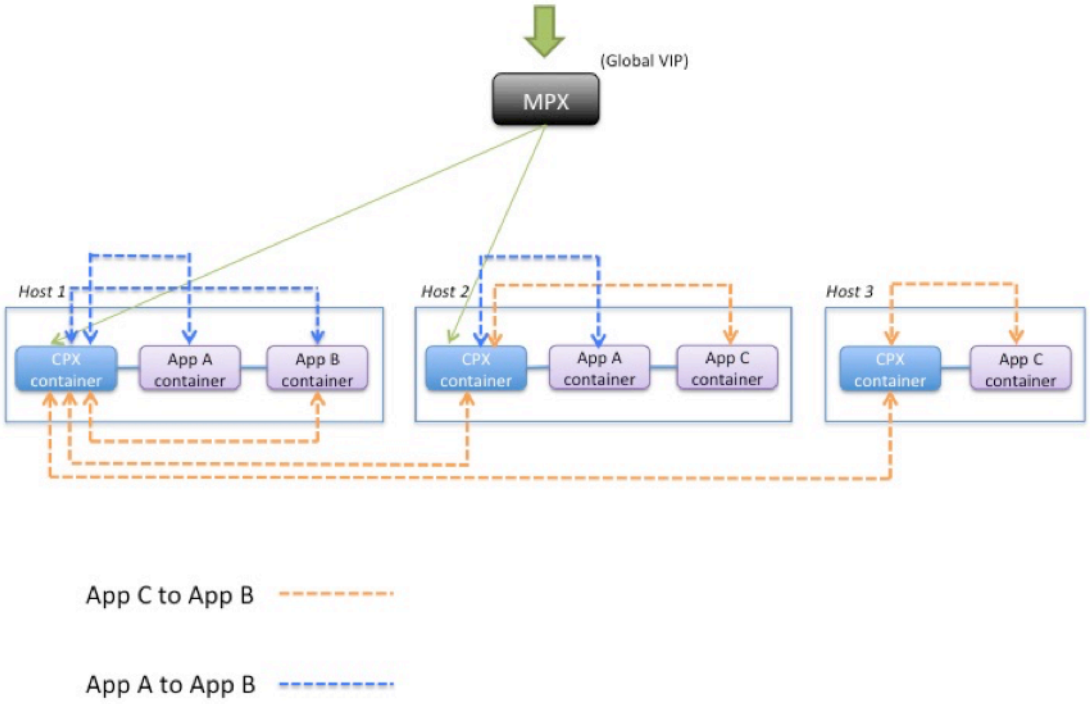
```
1      add cs vserver <name> <serviceType> 127.0.0.1 <port>
2
3      add cs vserver testcsvserver HTTP 127.0.0.1 30000
4  <!--NeedCopy-->
```


East-West トラフィックフローを可能にするための **NetScaler CPX** のプロキシとしての展開

November 23, 2023

この展開では、NetScaler CPX インスタンスは、複数のホスト上にあるアプリケーションコンテナ間の通信を可能にするためのプロキシとして機能します。マルチホストでアプリケーションと一緒にプロビジョニングされた NetScaler CPX インスタンスが、通信の最短パスを提供します。

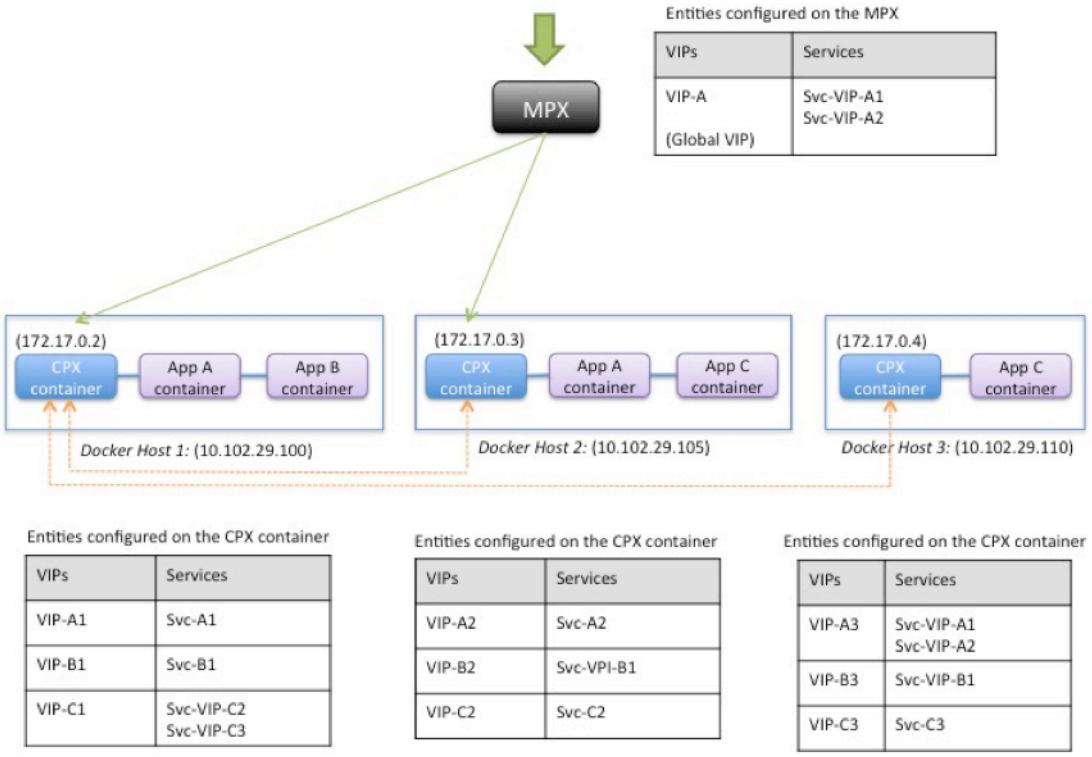
次の図は、NetScaler CPX インスタンスを介した 2 つのアプリケーション間のトラフィックフローを示します。



この図は、アプリケーション C とアプリケーション B の間のトラフィックフローと、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（いずれかのホストにある）が B に要求を送信すると、その要求は最初にアプリ C と同じホストにある NetScaler CPX コンテナで受信されます。次に、その NetScaler CPX コンテナが、アプリ B と同じホストでホストされている NetScaler CPX コンテナにトラフィックを渡すと、その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 に要求を送信するときもトラフィックは同じような流れになります。

この例では、グローバル VIP を介した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、NetScaler MPX も展開されています。NetScaler MPX からのトラフィックを NetScaler CPX コンテナが受信し、そこからアプリケーションコンテナにトラフィックが分散されます。

次の図は、このトポロジと、通信を可能にするために設定が必要な構成を示しています。



次の表は、この構成例の NetScaler CPX インスタンスに構成されている、IP アドレスとポートの一覧です。

Docker Host 1		Docker Host 2		Docker Host 3	
VIPs	**Services Bound to the VIP**	**VIPs**	**Services Bound to the VIP**	**VIPs**	**Services Bound to the VIP**
VIP-A1 172.17.0.2:30000	SVC-A1 10.102.29.100:80	VIP-A2 172.17.0.3:30000	SVC-A2 10.102.29.105:80	VIP-A3 172.17.0.4:30000	SVC-VIP-A1 10.102.29.100:30000
					SVC-VIP-A2 10.102.29.105:30000
VIP-B1 172.17.0.2:30001	SVC-B1 10.102.29.100:90	VIP-B2 172.17.0.3:30001	SVC-VIP-B1 10.102.29.100:30001	VIP-B3 172.17.0.4:30001	SVC-VIP-B1 10.102.29.100:30001
VIP-C1 172.17.02:30002	SVC-VIP-C2 10.102.29.105:30002	VIP-C2 172.17.0.3:30002	SVC-C2 10.102.29.105:70	VIP-C3 172.17.0.4:30002	SVC-C3 10.102.29.110:70
	SVC-VIP-C3 10.102.29.110:30002				

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間に、Linux Shell 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 30000-30002: 30000-30002 --
   ulimit core=-1 --privileged=truecpx:6.2
<!--NeedCopy-->
```

NetScaler ADM 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 の NetScaler CPX インスタンスで実行:

```
    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
<!--NeedCopy-->
```

Docker ホスト 2 の NetScaler CPX インスタンスで実行:

```
    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
<!--NeedCopy-->
```

Docker ホスト 3 の NetScaler CPX インスタンスで実行:

```
    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
<!--NeedCopy-->
```


単一ホストネットワークでの **NetScaler CPX** の展開

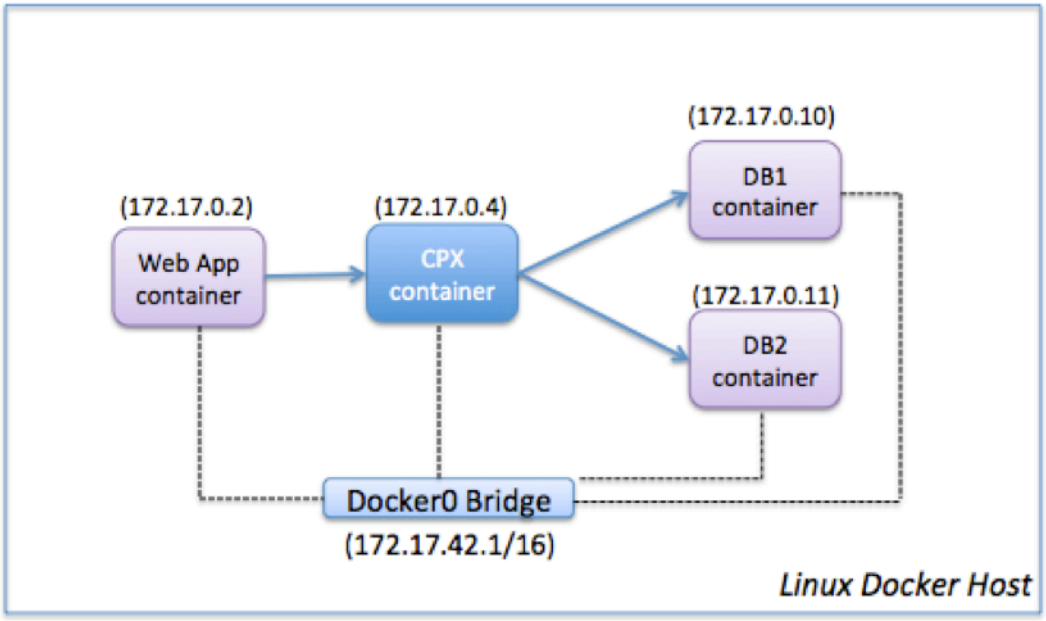
November 23, 2023

単一ホストネットワークでは、NetScaler CPX インスタンスは、同一ホストにある複数のアプリケーションコンテナ間のプロキシとして機能します。NetScaler CPX インスタンスのこの機能により、コンテナベースのアプリケーションのスケーラビリティとセキュリティが強化されます。このほかにも、パフォーマンスの最適化や、利用統計情報の理解を深めるためにも役立ちます。

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 インスタンスが、同じ Docker ホストのコンテナとして展開されます。コンテナはすべて docker0 ブリッジで接続されます。

この環境で、NetScaler CPX インスタンスは、同一の Docker ホストでコンテナとしてプロビジョニングされているアプリケーションのプロキシとして機能します。

次の図は、シングルホストトポロジを示しています。



この例では、Web アプリケーションコンテナ（172.17.0.2）がクライアントで、DB1（172.17.0.10）と DB2（172.17.0.11）の 2 つのデータベースコンテナがサーバーです。プロキシとして機能する NetScaler CPX コンテナ（172.17.0.4）が、クライアントとサーバーの間に配置されています。

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、最初に、NetScaler CPX コンテナで、2 台のサーバーを表す 2 つのサービスを構成する必要があります。次に、NetScaler CPX の IP アドレスと標準以外の HTTP ポート（81 など）を使用して仮想サーバーを構成します。標準以外のポートを使用するのは、NetScaler CPX で、標準 HTTP ポート 80 が Nitro との通信用に予約されているためです。

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にあるため、NAT 規則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 ADM のジョブ機能または 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
<!--NeedCopy-->
```

マルチホストネットワークでの **NetScaler CPX** の展開

November 23, 2023

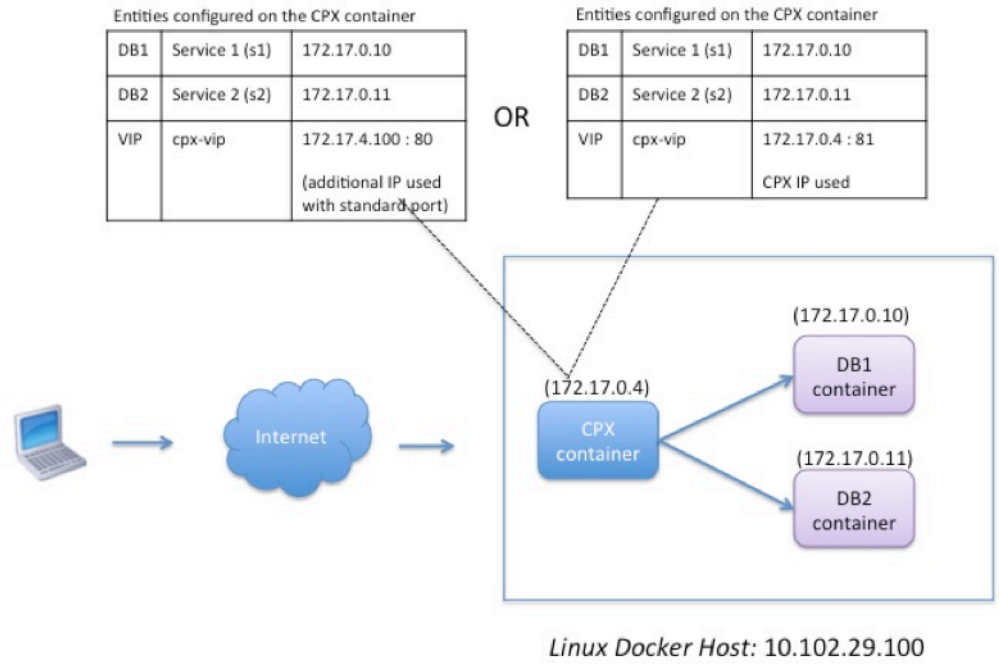
データセンター内の実稼働環境では、マルチホストネットワークに展開した NetScaler CPX インスタンスを構成して、負荷分散機能を使用できます。さらに、機能や分析データを監視することもできます。

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、別々のホストに展開されます。マルチホストのトポロジを実稼働環境で使用することで、NetScaler CPX インスタンスが一連のコンテナベースのアプリケーション、サーバー、さらには物理サーバーの負荷を分散します。

トポロジ **1**：**NetScaler CPX** とバックエンドサーバーが同一ホスト、クライアントが別のネットワークにある場合

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 ホストで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の別の場所から発信されます。このトポロジを実稼働環境で使用して、NetScaler CPX インスタンスが一連のコンテナベースのアプリケーションやサーバーの負荷を分散するようにできます。

次の図は、このトポロジを示しています。



この例では、NetScaler CPX インスタンス（172.17.0.4）と 2 台のサーバー（DB1（172.17.0.10）および DB2（172.17.0.11））は、IP アドレス 10.102.29.100 の同じ Docker ホストでプロビジョニングされています。クライアントは、ネットワークの他の場所にあります。

インターネットからのクライアント要求は、NetScaler CPX インスタンスに構成されている VIP で受信されます。そこから、要求が 2 台のサーバーに分散されます。

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 で追加 IP アドレスと標準ポートを使用する

1. 追加の IP アドレスを使用して NetScaler CPX コンテナの VIP を構成します。
2. Docker ホストに追加の IP アドレスを構成します。
3. 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 アドレスに転送する、NAT 規則を構成します。
4. 2 台のサーバーを NetScaler CPX インスタンスのサービスとして構成します。
5. 最後に、これらのサービスを VIP にバインドします。

この構成例では、10.x.x.x ネットワークはパブリックネットワークを表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 ADM のジョブ機能または NITRO API を使用して次のコマンドを実行します。

```
1     add service s1 172.17.0.10 HTTP 80
```



```
2      add service s2 172.17.0.11 HTTP 80
3      add lb vserver cpx-vip HTTP 172.17.4.100 80
4      bind lb vserver cpx-vip s1
5      bind lb vserver cpx-vip s2
6  <!--NeedCopy-->
```

Linux Shell プロンプトで次のコマンドを実行して、Docker ホストの追加のパブリック IP アドレスと NAT 規則を構成します:

```
1      ip addr add 10.102.29.103/24 dev eth0
2      iptables -t nat -A PREROUTING -p ip  -d 10.102.29.103 -j DNAT --to-
          destination 172.17.4.100
3  <!--NeedCopy-->
```

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

1. VIP と 2 つのサービスを NetScaler CPX インスタンスに構成します。この VIP では、標準以外のポート（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 規則を構成します。

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間に、Linux Shell プロンプトで次のコマンドを実行します:

```
1      docker run -dt -p 22 -p 80 -p 161/udp -p 50000:81 --ulimit core=-1
          --privileged=true cpx:6.2
2
3  <!--NeedCopy-->
```

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

```
1      add service s1 172.17.0.10 http 80
2      add service s2 172.17.0.11 http 80
3      add lb vserver cpx-vip HTTP 172.17.0.4 81
4      bind lb vserver cpx-vip s1
5      bind lb vserver cpx-vip s2
6  <!--NeedCopy-->
```

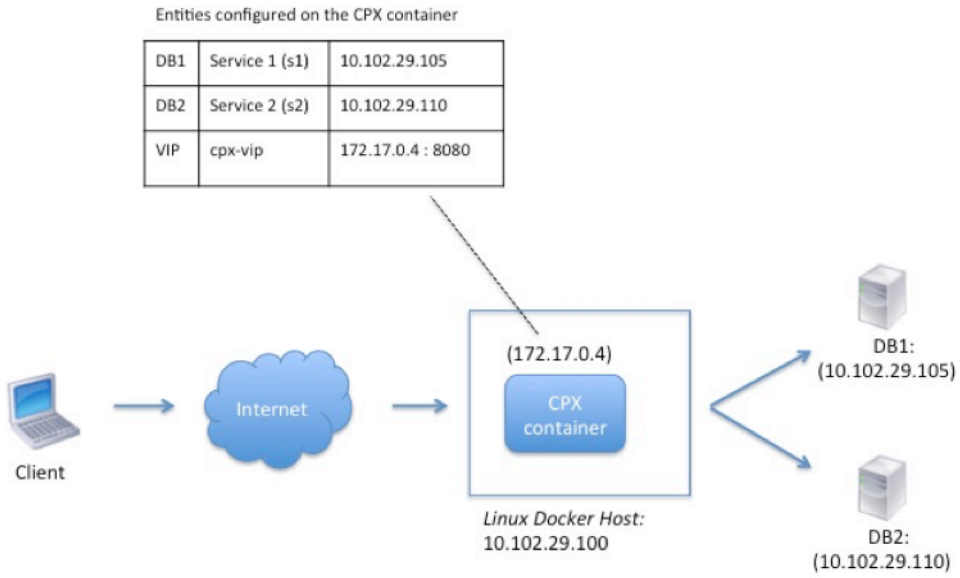
> 注:
>
> NetScaler CPX インスタンスのプロビジョニング中にポートマッピングを構成しなかった場合は、Linux Shell プロンプトで次のコマンドを実行して NAT 規則を構成します:
>
> **iptables -t nat -A PREROUTING -p tcp -m addrtype** –**dst-type LOCAL -m tcp** –**dport** 50000 **-j DNAT** –**to-destination**172.17.0.4:81

トポロジ **2**：**NetScaler CPX** と物理サーバー/クライアントが別々に配置されている場合

このトポロジでは、NetScaler CPX インスタンスのみが Docker ホストで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、ネットワークの別の場所にあります。

この環境では、物理サーバーのトラフィックを負荷分散するように、NetScaler CPX インスタンスを構成できます。

次の図は、このトポロジを示しています。



この例では、プロキシとして機能する 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配置されます。DB1（10.102.29.105）と DB2（10.102.29.110）の 2 台のサーバーは、ネットワーク上の Docker ホスト以外の場所にあります。インターネットからのクライアント要求は、NetScaler CPX インスタンスで受信されます。そこから、要求が 2 台のサーバーに分散されます。

NetScaler CPX を介したこのクライアントとサーバー間の通信を可能にするには、最初に、NetScaler CPX コンテナの作成中にポートマッピングを構成する必要があります。次に、NetScaler CPX コンテナに、2 台のサーバーを表す 2 つのサービスを構成します。最後に、NetScaler CPX IP アドレスと、マッピングされた非標準 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表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 コンテナを作成する間に、Linux Shell 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
<!--NeedCopy-->
```

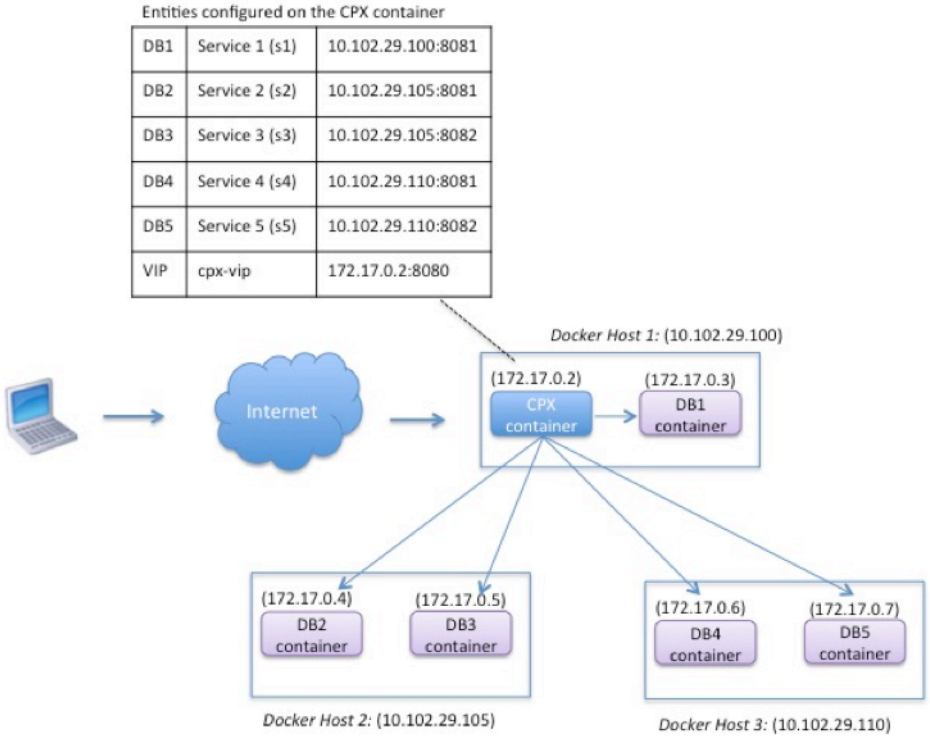
次に、NetScaler ADM のジョブ機能または NITRO API を使用して、次のコマンドを実行します。

```
    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
<!--NeedCopy-->
```

トポロジ **3**：**NetScaler CPX** とサーバーが別々のホストでプロビジョニングされている場合

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別々の Docker ホストで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。このトポロジを実稼働環境で使用して、NetScaler CPX インスタンスが一連のコンテナベースのアプリケーションやサーバーの負荷を分散するようにできます。

次の図は、このトポロジを示しています。



この例で、NetScaler CPX インスタンスと 1 台のサーバー（DB1）は、IP アドレス 10.102.29.100 の同じ Docker

ホストでプロビジョニングされています。その他 4 台のサーバー（DB2、DB3、DB4、および DB5）は、2 つの異なる Docker ホスト（10.102.29.105 および 10.102.29.110）でプロビジョニングされています。

インターネットからのクライアント要求は NetScaler CPX インスタンスで受信され、そこから、要求が 5 台のサーバーに分散されます。この通信を有効にするには、次のように構成する必要があります:

1. NetScaler CPX コンテナの作成中にポートマッピングを設定します。この例では、トラフィックをコンテナのポート 8080 からホストのポート 8080 に転送する必要があります。クライアント要求がホストのポート 8080 に到達すると、CPX コンテナのポート 8080 にマッピングされます。
2. 5 台のサーバーを NetScaler CPX インスタンスの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、該当する 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け取る NetScaler CPX インスタンスの 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 で表します。
4. 最後に、これらのサービスを VIP にバインドします。

この構成例では、10.x.x.x ネットワークはパブリックネットワークを表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 コンテナを作成する間に、Linux Shell プロンプトで次のコマンドを実行します:

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

NetScaler ADM 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。

```
1     add service s1 10.102.29.100 HTTP 8081
2     add service s2 10.102.29.105 HTTP 8081
3     add service s3 10.102.29.105 HTTP 8082
4     add service s4 10.102.29.110 HTTP 8081
5     add service s5 10.102.29.110 HTTP 8082
6     add lb vserver cpx-vip HTTP 172.17.0.2 8080
7     bind lb vserver cpx-vip s1
8     bind lb vserver cpx-vip s2
9     bind lb vserver cpx-vip s3
10    bind lb vserver cpx-vip s4
11    bind lb vserver cpx-vip s5
12 <!--NeedCopy-->
```

ネットワークに直接アクセスできる **NetScaler CPX** を導入

November 23, 2023

ブリッジネットワークモードでは、NetScaler CPX インスタンスがネットワークに直接アクセス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、ネットワークポートを docker0 ブリッジにバインドします。

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

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表しています。

このシナリオを構成するには、Linux シェルプロンプトで次のコマンドを実行します:

```
1      ip addr add 10.102.29.100/24 dev docker0;
2      ip addr del 10.102.29.100/24 dev eth0;
3      brctl addif docker0 eth0;
4      ip route del default;
5      ip route add default via 10.102.29.1 dev docker0
6  <!--NeedCopy-->
```

NetScaler ADM 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。

```
1      add service s1 172.17.0.8 http 80
2      add service s2 172.17.0.9 http 80
3      add lb vserver cpx-vip HTTP 10.102.29.102 80
4      bind lb vserver cpx-vip s1
5      bind lb vserver cpx-vip s2
6  <!--NeedCopy-->
```

ConfigMaps を使用した Kubernetes での NetScaler CPX の構成

November 23, 2023

Kubernetes では、ConfigMaps を使用して NetScaler CPX インスタンスを構成できます。ConfigMaps を使用すると、インスタンスの起動時に 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する NetScaler 固有の構成および 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タグの下で、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タグの下で、NetScaler CPX インスタンスで実行するすべてのシェルコマンドを追加する必要があります。

> 重要:
>
> - このタグは、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「#」の文字を追加してください。
> - タグでは大文字と小文字を区別しません。
> - NetScaler CPX コンテナを構成ファイルとともに展開する際に障害が発生すると、それらの障害は`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 を変更すると、更新された構成は NetScaler CPX インスタンスが再起動された時にのみ適用されます。

次に、構成ファイルの例を示します:

```
#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
<!--NeedCopy-->
```

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
<!--NeedCopy-->
```

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。この ConfigMap は、NetScaler CPX インスタンスの展開で使用される YAML ファイルで使用できます。

作成した 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`

例:

```
    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
<!--NeedCopy-->
```

作成した ConfigMap は、`cpx-config` NetScaler CPX インスタンスのデプロイに使用される YAML ファイルに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"
```



```
30         - name: "NS_ROUTABLE"
31           value: "FALSE"
32         - name: "KUBERNETES_TASK_ID"
33           valueFrom:
34             fieldRef:
35               fieldPath: metadata.name
36       imagePullPolicy: Never
37     volumes:
38     - name: config-volume
39       configMap:
40         name: cpx-config
41 <!--NeedCopy-->
```

NetScaler CPX が展開され、ConfigMap に指定された構成を開始すると、NetScaler CPX インスタンスに`cpx-config`が適用されます。

Kubernetes ノードのローカル DNS キャッシュとして NetScaler CPX を導入

November 23, 2023

Kubernetes クラスター内のアプリケーションポッドは、DNS を利用して他のアプリケーションポッドと通信します。Kubernetes クラスター内のアプリケーションからの DNS 要求は、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が広く採用されているため、Kubernetes クラスター内の DNS 要求レートは増加しています。その結果、Kubernetes DNS（kube-dns）が過負荷になります。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 要求を NetScaler CPX に転送できるようになりました。結果として、DNS 要求をより迅速に解決し、Kubernetes DNS の負荷を大幅に減らすことができます。

NetScaler CPX をデプロイするには、Kubernetes DaemonSet エンティティを使用して、Kubernetes クラスター内の各ノードで NetScaler CPX ポッドをスケジュールします。Kubernetes DaemonSet を使用すると、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が保証されます。
アプリケーションポッドから CPX DNS ポッドにトラフィックを転送するには、エンドポイントを NetScaler CPX ポッドとする 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 解決に NetScaler CPX サービスクラスター IP アドレスを使用するようにするには、各ノードの kubelet 構成ファイルを NetScaler CPX サービスクラスター IP で更新する必要があります。

NetScaler CPX をノードローカル DNS キャッシュとして展開できるようにするために、次の環境変数が導入されました。

- `KUBE_DNS_SVC_IP`: NetScaler `kube-dns` CPX ポッドの構成をトリガーするための必須引数であるサービスのクラスター IP アドレスを指定します。NetScaler CPX ポッドは、DNS クエリ応答が NetScaler CPX キャッシュに存在しない場合に、この IP アドレスに DNS クエリを送信します。

- 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、ノードでローカル DNS を設定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送信される DNS 要求をノード内のローカル NetScaler iptables CPX ポッドに送信する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 を介して受信された場合でも、DNS 要求に TCP を使用するように強制します。
- NS_DNS_EXT_RESLV_IP: 特定のドメインの DNS 要求を送信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送信するために照合する外部ドメイン文字列を指定します。

NetScaler CPX を DNS キャッシュとしてノードにデプロイ

NetScaler CPX を Kubernetes クラスターのローカル DNS キャッシュとして展開するには、次のタスクが含まれます。

マスターノードの場合:

- エンドポイントを NetScaler CPX ポッドとして使用する Kubernetes サービスを作成
- NetScaler CPX ポッドの環境変数を定義するためのコンフィグマップの作成
- Kubernetes DaemonSet を使用して、Kubernetes クラスター内の各ノードで NetScaler CPX ポッドをスケジュールします。

ワーカーノードの場合:

- NetScaler CPX サービスのクラスター IP アドレスを使用して kubelet 構成ファイルを変更し、DNS リクエストを NetScaler CPX に転送します。

Kubernetes マスターノードでの構成

Kubernetes マスターノードで次の手順を実行して、NetScaler CPX をノードのローカル DNS キャッシュとして展開します。

1. 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。cpx_dns_svc.yaml

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

   cpx_dns_svc.yamlファイルは次のように指定されます:


```
        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コンフィグマップを作成します。これらの環境変数は、NetScaler CPX サービスと Kube DNS サービスの IP アドレスを渡すために使用されます。この手順では、ファイル内のデータ（キーと値のペア）として指定された環境変数を使用して、サンプルの 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、キーと値のペアとして環境変数を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、ConfigMap のサンプルです:

```
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. マスターノードで NetScaler CPX 用の Kubernetes デーモンセットを作成します。

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

cpx_daemonset.yamlファイルは次のように指定されます:

```
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
```



```
37          type: Directory
```

Kubernetes クラスターのワーカーノードでの構成

マスターノードでの構成が完了したら、ワーカーノードで次の手順を実行します。

1. 次のいずれかの手順を使用して、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、kubelet 構成ファイルを変更します。

   - ノードの kubelet を再構成する手順に従い、--cluster-dns引数の値を次の形式で変更します。

     ```
     1       --cluster-dns=<CPX_DNS_SVC_IP>,<KUBE_DNS_SVC_IP>
     ```

     または

   - 次の手順で/etc/systemd/system/kubelet.service.d/10-kubeadm.confファイルを編集し、--cluster-dns引数を変更します。

     a) kubelet 構成を編集し、引数に NetScaler kube-dns CPX サービスのクラスター IP アドレスとサービス IP アドレスを指定します。--cluster-dns

     ```
     1  root@node:~# cat /etc/systemd/system/kubelet.service.d/10-
        kubeadm.conf | grep KUBELET\_DNS\_ARGS
     2
     3  Environment="KUBELET_DNS_ARGS=--cluster-dns
        =10.111.95.145,10.96.0.10 --cluster-domain=cluster.
        local"
     4  ExecStart=/usr/bin/kubelet $KUBELET_KUBECONFIG_ARGS
        $KUBELET_CONFIG_ARGS $KUBELET_DNS_ARGS
     ```

     b) 次のコマンドを使用して、ノードの kubelet を再読み込みします。

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

Google Compute Engine での NetScaler CPX プロキシの展開

March 25, 2024

この導入ガイドでは、エンタープライズネットワーク内で実行されている NetScaler ADM を搭載した NetScaler CPX を Google Cloud の Google コンピュートエンジン（GCE）に Docker で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 ADM 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、レイヤー 7 機能、SSL オフロード、複数のプロトコル、NITRO API をフルサポートするコンテナベースのプロキシです。NetScaler ADM 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。NetScaler ADM はライセンスサーバーとして、オンプレミスまたはクラウドで実行される NetScaler CPX インスタンスの使用権限を提供します。

CPX と CPX Express は同じイメージです。NetScaler ADM を使用して CPX イメージのライセンスを取得してインストールすると、Docker App Store（リリース 11 または 12）の CPX イメージは完全な CPX インスタンスになります。ライセンスが付与されないと、CPX イメージは 20Mbps と 250 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 2GB のメモリと 1vCPU
- GCE から利用できる Docker オープンソース
- インターネットまたは GCE への VPN 接続を使用してオンプレミスで実行される NetScaler ADM

> 注
>
> NetScaler ADM をデプロイする方法については、「NetScaler ADM のデプロイ」を参照してください。

構成の手順

この展開を構成するには、以下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 インスタンスと通信するリモート API 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 ADM を通じて NetScaler CPX のライセンスを取得します。
6. NetScaler CPX で負荷分散サービスを構成し、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) NetScaler CPX を負荷分散するように構成し、両方の Web サービスへの負荷分散を確認します。

手順 **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、以下の例に示したコマンドを使用して、その 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/mlvEqgnyTQy+e6oEYN2Y2kqXceUhXagThnqCoxcEJ3+KM4R
mYdoe/BJ/J/6rHOjq7Omk24z2qB3RU1uAv57iY5VGw5p45uZB4C4pNNsBJXoCvPn
TGAs/7IrekFZDDgVraPx/hdiwopQ8NltSfZCyu/jPpWFK28TR8yfVlzYFwibj5WK
dHM7ZTqlA1tHIG+agyPf3Rae0jPMsHR6q+arXVwMccyOi+ULU0z8mHUJ3iEMIrpT
X+80KaN/ZjibfsBOCjcfiJSB/acn4nxQQgNZigna32velafhQivsNREFeJpzENiG
HOoyC6qVeOgKrRiKxzymj0FIMLru/iFF5pSWcBQB7PYlt8J0G80lAcPr6VCiN+4c
NKv03SdvA69dCOj79PuO9IIvQsJXsSq96HB+TeEmmL+xSdpGtGdCJHHM1fDeCqkZ
hT+RtBGQL2SEdWjxbF43oQopocT8cHvyX6Zaltn0svoGs+wX3Z/H6/8P5anog43U
65c0A+64Jj00rNDr8j31izhtQMRo892kGeQAaaxg4Pz6HnS7hRC+cOMHUU4HA7iM
zHrouAdYeTZeZEQOA7SxtCME9ZnGwe2grxPXh/U/80WJGkzLFNcTKdv+rwARAQAB
tDdEb2NrZXIgUmVsZWFzZSBUb29sIChyZWxlYXNlZG9ja2VyKSA8ZG9ja2VyQGRv
Y2tlci5jb20+iQIcBBABCgAGBQJWw7vdAAoJEFyzYeVS+w0QHysP/i37m4SyoOCV
cnybl18vzwBEcp4VCRbXvHvOXty1gccVIV8/aJqNKgBV97lY3vrpOyiIeB8ETQeg
srxFE7t/Gz0rsLObqfLEHdmn5iBJRkhLfCpzjeOnyB3Z0IJB6UogO/msQVYe5CXJ
l6uwr0AmoiCBLrVlDAktxVh9RWch0l0KZRX2FpHu8h+uM0/zySqIidlYfLa3y5oH
scU+nGU1i6ImwDTD3ysZC5jp9aVfvUmcESyAb4vvdcAHR+bXhA/RW8QHeeMFliWw
7Z2jYHyuHmDnWG2yUrnCqAJTrWV+OfKRIzzJFBs4e88ru5h2ZIXdRepw/+COYj34
LyzxR2cxr2u/xvxwXCkSMe7F4KZAphD+1ws61FhnUMi/PERMYfTFuvPrCkq4gyBj
t3fFpZ2NR/fKW87QOeVcn1ivXl9id3MMs9KXJsg7QasT7mCsee2VIFsxrkFQ2jNp
D+JAERRn9Fj4ArHL5TbwkkFbZZvSi6fr5h2GbCAXIGhIXKnjjorPY/YDX6X8AaHO
W1zblWy/CFr6VFl963jrjJgag0G6tNtBZLrclZgWhOQpeZZ5Lbvz2ZA5CqRrfAVc
wPNW1fObFIRtqV6vuVluFOPCMAAnOnqR02w9t17iVQjO3oVN0mbQi9vjuExXh1Yo
ScVetiO6LSmlQfVEVRTqHLMgXyR/EMo7iQIcBBABCgAGBQJXSWBlAAoJEFyzYeVS
+w0QeH0QAI6btAfYwYPuAjfRUy9qlnPhZ+xt1rnwsUzsbmo8K3XTNh+l/R08nu0d
sczw30Q1wju28fh1N8ay223+69f0+yICaXqR18AbGgFGKX7vo0gfEVaxdItUN3eH
NydGFzmeOKbAlrxIMECnSTG/TkFVYO9Ntlv9vSN2BupmTagTRErxLZKnVsWRzp+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]
49 Get:4 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/restricted
      Sources [5,376 B]
50 Get:5 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/multiverse
      Sources [181 kB]
51 Get:6 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/universe
      Sources [8,044 kB]
52 Get:7 http://archive.canonical.com/ubuntu yakkety InRelease [11.5 kB]
53 Get:8 http://security.ubuntu.com/ubuntu yakkety-security InRelease [102
       kB]
54 Get:9 https://apt.dockerproject.org/repo ubuntu-yakkety InRelease [47.3
       kB]
55 Get:10 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/main
      Sources [903 kB]
56 Get:11 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
      restricted Sources [2,688 B]
57 Get:12 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
      universe Sources [57.9 kB]
58 Get:13 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
      multiverse Sources [3,172 B]
59 Get:14 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
      main Sources [107 kB]
60 Get:15 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
      main amd64 Packages [268 kB]
61 Get:16 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
      main Translation-en [122 kB]
62 Get:17 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
      universe amd64 Packages [164 kB]
63 Get:18 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
      universe Translation-en [92.4 kB]
64 Get:19 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
      multiverse amd64 Packages [4,840 B]
65 Get:20 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-updates/
      multiverse Translation-en [2,708 B]
66 Get:21 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports/
      universe Sources [2,468 B]
67 Get:22 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports/
      main Sources [2,480 B]
68 Get:23 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports/
      main amd64 Packages [3,500 B]
69 Get:24 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports/
      universe amd64 Packages [3,820 B]
70 Get:25 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety-backports/
      universe Translation-en [1,592 B]
71 Get:26 http://archive.canonical.com/ubuntu yakkety/partner amd64
      Packages [2,480 B]
72 Get:27 http://security.ubuntu.com/ubuntu yakkety-security/main Sources
      [47.7 kB]
73 Get:28 https://apt.dockerproject.org/repo ubuntu-yakkety/main amd64
      Packages [2,453 B]
74 Get:29 http://security.ubuntu.com/ubuntu yakkety-security/universe
      Sources [20.7 kB]
75 Get:30 http://security.ubuntu.com/ubuntu yakkety-security/multiverse
```



```
      Sources [1,140 B]
76  Get:31 http://security.ubuntu.com/ubuntu yakkety-security/restricted
      Sources [2,292 B]
77  Get:32 http://security.ubuntu.com/ubuntu yakkety-security/main amd64
      Packages [150 kB]
78  Get:33 http://security.ubuntu.com/ubuntu yakkety-security/main
      Translation-en [68.0 kB]
79  Get:34 http://security.ubuntu.com/ubuntu yakkety-security/universe
      amd64 Packages [77.2 kB]
80  Get:35 http://security.ubuntu.com/ubuntu yakkety-security/universe
      Translation-en [47.3 kB]
81  Get:36 http://security.ubuntu.com/ubuntu yakkety-security/multiverse
      amd64 Packages [2,832 B]
82  Fetched 10.8 MB in 2s (4,206 kB/s)
83  Reading package lists... Done
84  Reading package lists...
85  Building dependency tree...
86  Reading state information...
87  The following additional packages will be installed:
88  aufs-tools cgroupfs-mount libltdl7
89  The following NEW packages will be installed:
90  aufs-tools cgroupfs-mount docker-engine libltdl7
91  0 upgraded, 4 newly installed, 0 to remove and 37 not upgraded.
92  Need to get 21.2 MB of archives.
93  After this operation, 111 MB of additional disk space will be used.
94  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]
95  Get:2 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/universe
      amd64 cgroupfs-mount all 1.3 [5,778 B]
96  Get:3 http://us-west1.gce.archive.ubuntu.com/ubuntu yakkety/main amd64
      libltdl7 amd64 2.4.6-1 [38.6 kB]
97  Get:4 https://apt.dockerproject.org/repo ubuntu-yakkety/main amd64
      docker-engine amd64 17.05.0~ce-0~ubuntu-yakkety [21.1 MB]
98  Fetched 21.2 MB in 1s (19.8 MB/s)
99  Selecting previously unselected package aufs-tools.
100 (Reading database ... 63593 files and directories currently installed.)
101 Preparing to unpack .../aufs-tools_1%3a3.2+20130722-1.1ubuntu1_amd64.
      deb ...
102 Unpacking aufs-tools (1:3.2+20130722-1.1ubuntu1) ...
103 Selecting previously unselected package cgroupfs-mount.
104 Preparing to unpack .../cgroupfs-mount_1.3_all.deb ...
105 Unpacking cgroupfs-mount (1.3) ...
106 Selecting previously unselected package libltdl7:amd64.
107 Preparing to unpack .../libltdl7_2.4.6-1_amd64.deb ...
108 Unpacking libltdl7:amd64 (2.4.6-1) ...
109 Selecting previously unselected package docker-engine.
110 Preparing to unpack .../docker-engine_17.05.0~ce-0~ubuntu-yakkety_amd64
      .deb ...
111 Unpacking docker-engine (17.05.0~ce-0~ubuntu-yakkety) ...
112 Setting up aufs-tools (1:3.2+20130722-1.1ubuntu1) ...
113 Processing triggers for ureadahead (0.100.0-19) ...
114 Setting up cgroupfs-mount (1.3) ...
115 Processing triggers for libc-bin (2.24-3ubuntu2) ...
```



```
116 Processing triggers for systemd (231-9ubuntu4) ...
117 Setting up libltdl7:amd64 (2.4.6-1) ...
118 Processing triggers for man-db (2.7.5-1) ...
119 Setting up docker-engine (17.05.0~ce-0~ubuntu-yakkety) ...
120 Created symlink /etc/systemd/system/multi-user.target.wants/docker.
       service → /lib/systemd/system/docker.service.
121 Created symlink /etc/systemd/system/sockets.target.wants/docker.socket
       → /lib/systemd/system/docker.socket.
122 Processing triggers for ureadahead (0.100.0-19) ...
123 Processing triggers for libc-bin (2.24-3ubuntu2) ...
124 Processing triggers for systemd (231-9ubuntu4) ...
125 + sudo -E sh -c docker version
126 Client:
127 Version: 17.05.0-ce
128 API version: 1.29
129 Go version: go1.7.5
130 Git commit: 89658be
131 Built: Thu May 4 22:15:36 2017
132 OS/Arch: linux/amd64
133
134 Server:
135 Version: 17.05.0-ce
136 API version: 1.29 (minimum version 1.12)
137 Go version: go1.7.5
138 Git commit: 89658be
139 Built: Thu May 4 22:15:36 2017
140 OS/Arch: linux/amd64
141 Experimental: false
142
143 If you would like to use Docker as a non-root user, you should now
       consider
144 adding your user to the "docker" group with something like:
145
146 sudo usermod -aG docker albert_lee
147
148 Remember that you will have to log out and back in for this to take
       effect.
149
150 WARNING: Adding a user to the "docker" group will grant the ability to
       run
151 containers which can be used to obtain root privileges on the
152 docker host.
153 Refer to https://docs.docker.com/engine/security/security/#docker-
       daemon-attack-surface
154 for more information.
155
156 $
157
158 \*\*$ sudo docker info\*\*
159 Containers: 0
160 Running: 0
161 Paused: 0
162 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
$
<!--NeedCopy-->
```


手順 **2**: **Docker** インスタンスと通信するリモート **API** を構成する

Docker インスタンスと通信する API 用に 4243 ポートを開放します。このポートは、NetScaler ADM 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","GoVersion":"go1.7.5","Os":"linux","Arch":"
    amd64","KernelVersion":"4.8.0-52-generic","BuildTime":"2017-05-04
    T22:15:36.071254972+00:00" }

(external)$

<!--NeedCopy-->
```

手順 **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 ADM を使用して CPX イメージのライセンスを取得してインストールすると、イメージは 1 Gbps のパフォーマンスを備えた完全な CPX インスタンスになります。ライセンスが付与されないと、イメージは 20Mbps と 250 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: sha256:31
    a65cfa38833c747721c6fbc142faec6051e5f7b567d8b212d912b69b4f1ebe
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
$
<!--NeedCopy-->
```

ステップ **4: NetScaler CPX** インスタンスを作成する

NetScaler CPX イメージを Docker ホストにインストールします。次の例のように特定のサービスのポートを開き、NetScaler ADM 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,
```



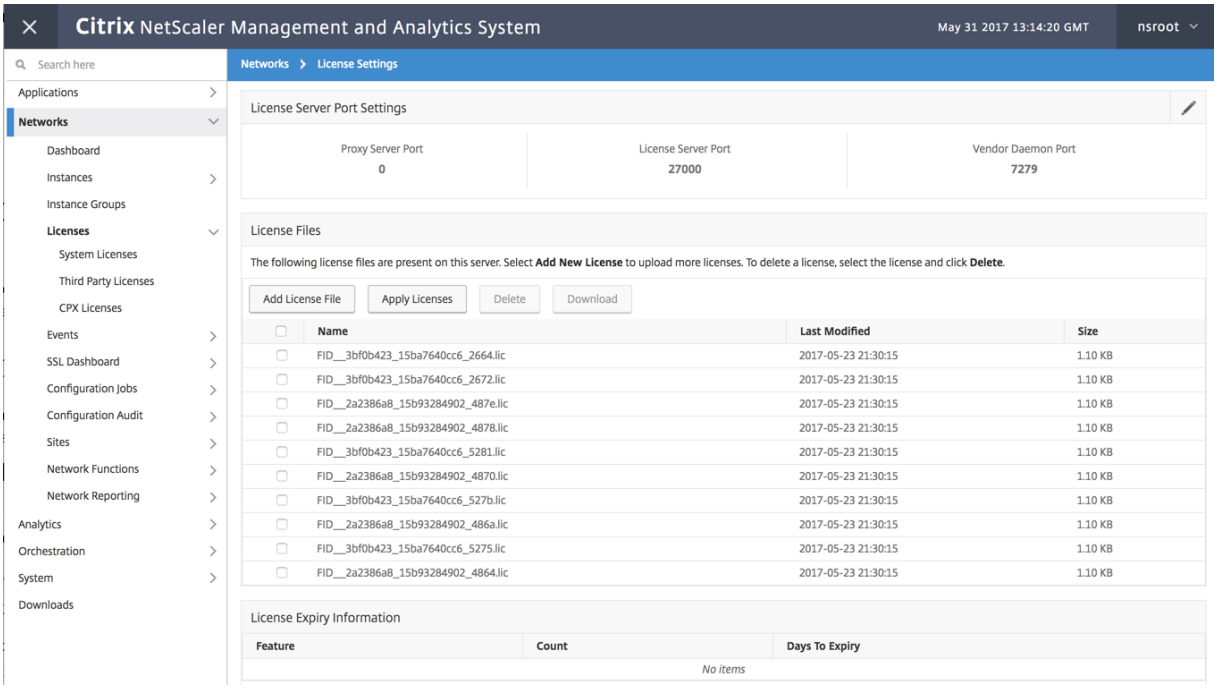
```
    0.0.0.0:5443->443/tcp gifted_perlman
$

$ \*\*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
```

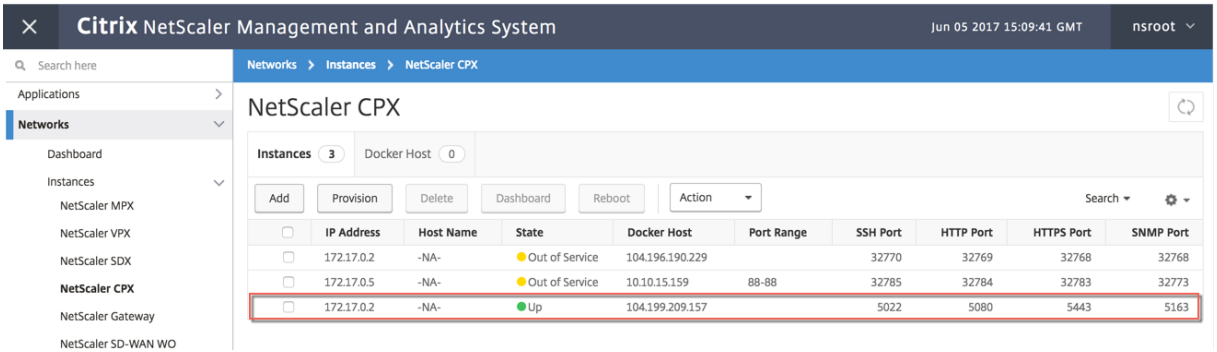


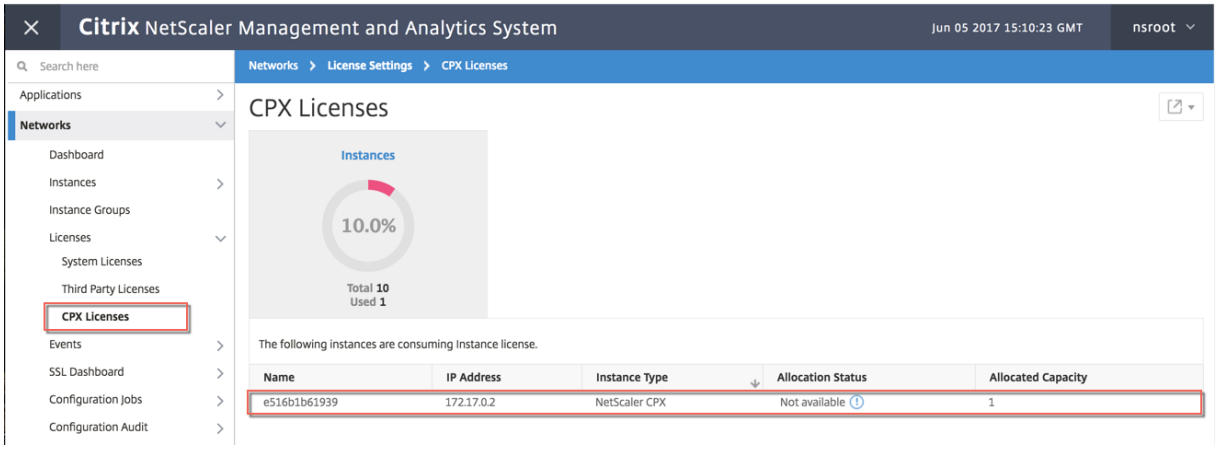
```
DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5443 to:172.17.0.2:443
DNAT udp -- 0.0.0.0/0 0.0.0.0/0 udp dpt:5163 to:172.17.0.2:161
DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:50000 to:172.17.0.2:88
DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5080 to:172.17.0.2:80
DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5022 to:172.17.0.2:22
$
<!--NeedCopy-->
```

ステップ **5: NetScaler ADM** を通じて **NetScaler CPX** のライセンスを取得する

NetScaler ADM 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 ADM と通信して情報を送信していることを確認できるはずです。次の画像は、NetScaler CPX が NetScaler ADM からライセンスを取得しているところを示しています。







手順 **6**: **NetScaler CPX** で負荷分散サービスを構成し、その構成を確認する

はじめに、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 サーバーに負荷を分散し、その構成をテストします。

NGINX Web サーバーをインストールする　以下の例に示したコマンドを使用して、NGINX Web サーバーをインストールします。

```
$ sudo docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
Digest: sha256:41
    ad9967ea448d7c2b203c699b429abe1ed5af331cd92533900c6d77490e0268
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
```



```
27
28 Chain POSTROUTING (policy ACCEPT)
29 target prot opt source destination
30 MASQUERADE all -- 172.17.0.0/16 0.0.0.0/0
31 MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:443
32 MASQUERADE udp -- 172.17.0.2 172.17.0.2 udp dpt:161
33 MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:88
34 MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:80
35 MASQUERADE tcp -- 172.17.0.2 172.17.0.2 tcp dpt:22
36 MASQUERADE tcp -- 172.17.0.3 172.17.0.3 tcp dpt:80
37 MASQUERADE tcp -- 172.17.0.4 172.17.0.4 tcp dpt:80
38
39 Chain DOCKER (2 references)
40 target prot opt source destination
41 RETURN all -- 0.0.0.0/0 0.0.0.0/0
42 DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5443 to:172.17.0.2:443
43 DNAT udp -- 0.0.0.0/0 0.0.0.0/0 udp dpt:5163 to:172.17.0.2:161
44 DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:50000 to:172.17.0.2:88
45 DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5080 to:172.17.0.2:80
46 DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:5022 to:172.17.0.2:22
47 DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:81 to:172.17.0.3:80
48 DNAT tcp -- 0.0.0.0/0 0.0.0.0/0 tcp dpt:82 to:172.17.0.4:80
49 $
50 <!--NeedCopy-->
```

NetScaler CPX による負荷分散の構成と両方の **Web** サービスへの負荷分散の検証

```
1 $ \*\*ssh -p 5022 root@localhost\*\*
2 root@localhost's password:
3 Welcome to nsoslx 1.0 (GNU/Linux 4.8.0-52-generic x86_64)
4
5 * Documentation: https://www.citrix.com/
6 Last login: Mon Jun 5 18:58:54 2017 from 172.17.0.1
7 root@44ca1c6c0907:~#
8 root@44ca1c6c0907:~#
9 root@44ca1c6c0907:~#
10 root@44ca1c6c0907:~#
11 root@44ca1c6c0907:~# \*\*cli_script.sh "add service web1 172.17.0.3
      HTTP 80"\*\*
12 exec: add service web1 172.17.0.3 HTTP 80
13 Done
14 root@44ca1c6c0907:~# \*\*cli_script.sh "add service web2 172.17.0.4
      HTTP 80"\*\*
15 exec: add service web2 172.17.0.4 HTTP 80
16 Done
17 root@44ca1c6c0907:~# \*\*cli_script.sh "add lb vserver cpx-vip HTTP
      172.17.0.2 88"\*\*
18 exec: add lb vserver cpx-vip HTTP 172.17.0.2 88
19 Done
20 root@44ca1c6c0907:~# \*\*cli_script.sh "bind lb vserver cpx-vip web1
      "\*\*
21 exec: bind lb vserver cpx-vip web1
22 Done
23 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 {
```



```
75
76 width: 35em;
77 margin: 0 auto;
78 font-family: Tahoma, Verdana, Arial, sans-serif;
79  }
80
81 \\</style\\>
82 \\</head\\>
83 \<body\>
84 \<h1\>Welcome to nginx\!\</h1\>
85 \<p\>If you see this page, the nginx web server is successfully
      installed and
86 working. Further configuration is required.\\</p\\>
87
88 \<p\>For online documentation and support please refer to
89 \\<a href="http://nginx.org/"\\>nginx.org\\</a\\>.\\<br/\\>
90 Commercial support is available at
91 \\<a href="http://nginx.com/"\\>nginx.com\\</a\\>.\\</p\\>
92
93 \<p\>\<em\>Thank you for using nginx.\</em\>\</p\>
94 \\</body\\>
95 \\</html\\>
96 (external)$
97
98
99 (external)$ for i in {
100  1..100 }
101   ; \*\*do curl http://104.199.209.157:50000 -o /dev/null ; done\*\*
102
103   % Total    % Received % Xferd  Average Speed   Time    Time     Time
      Current
104
105                                  Dload  Upload   Total   Spent    Left
      Speed
106
107 100   612  100   612    0     0   1767      0 --:--:-- --:--:--
      --:--:--  1768
108
109   % Total    % Received % Xferd  Average Speed   Time    Time     Time
      Current
110
111                                  Dload  Upload   Total   Spent    Left
      Speed
112
113 100   612  100   612    0     0   1893      0 --:--:-- --:--:--
      --:--:--  1894
114
115   % Total    % Received % Xferd  Average Speed   Time    Time     Time
      Current
116
117                                  Dload  Upload   Total   Spent    Left
      Speed
118
```



```
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
```



```
154
155 100   612  100   612    0     0   1802      0 --:--:-- --:--:--
       --:--:--  1800
156
157   % Total    % Received % Xferd  Average Speed   Time    Time     Time
       Current
158
159                                  Dload  Upload   Total   Spent    Left
       Speed
160
161 100   612  100   612    0     0   1902      0 --:--:-- --:--:--
       --:--:--  1906
162
163   % Total    % Received % Xferd  Average Speed   Time    Time     Time
       Current
164
165                                  Dload  Upload   Total   Spent    Left
       Speed
166
167 100   612  100   612    0     0   1843      0 --:--:-- --:--:--
       --:--:--  1848
168
169
170
171   % Total    % Received % Xferd  Average Speed   Time    Time     Time
       Current
172
173                                  Dload  Upload   Total   Spent    Left
       Speed
174
175 100   612  100   612    0     0   1862      0 --:--:-- --:--:--
       --:--:--  1860
176
177   % Total    % Received % Xferd  Average Speed   Time    Time     Time
       Current
178
179                                  Dload  Upload   Total   Spent    Left
       Speed
180
181 100   612  100   612    0     0   1806      0 --:--:-- --:--:--
       --:--:--  1810
182
183   % Total    % Received % Xferd  Average Speed   Time    Time     Time
       Current
184
185                                  Dload  Upload   Total   Spent    Left
       Speed
186
187 100   612  100   612    0     0   1702      0 --:--:-- --:--:--
       --:--:--  1704
188
189 (external)$
190
```



```
191
192
193
194
195 root@44ca1c6c0907:~# \*\*cli_script.sh 'stat lb vserver cpx-vip'\*\*
196
197 exec: stat lb vserver cpx-vip
198
199
200
201 Virtual Server Summary
202
203                       vsvrIP  port     Protocol        State   Health
    actSvcs
204
205 cpx-vip           172.17.0.2    88         HTTP           UP      100
            2
206
207
208
209            inactSvcs
210
211 cpx-vip            0
212
213
214
215 Virtual Server Statistics
216
217                                            Rate (/s)
    Total
218
219 Vserver hits                                      0
    101
220
221 Requests                                         0
    101
222
223 Responses                                        0
              101
224
225 Request bytes                                    0
    8585
226
227 Response bytes                                   0
    85850
228
229 Total Packets rcvd                               0
    708
230
231 Total Packets sent                               0
    408
232
233 Current client connections                      --
```



```
        0
234
235 Current Client Est connections                    --
        0
236
237 Current server connections                       --
                          0
238
239 Current Persistence Sessions                     --
        0
240
241 Requests in surge queue                          --
        0
242
243 Requests in vserver's surgeQ                     --
        0
244
245 Requests in service's surgeQs                    --
        0
246
247 Spill Over Threshold                             --
        0
248
249 Spill Over Hits                                  --
        0
250
251 Labeled Connection                               --
        0
252
253 Push Labeled Connection                          --
        0
254
255 Deferred Request                                  0
        0
256
257 Invalid Request/Response                         --
        0
258
259 Invalid Request/Response Dropped                 --
        0
260
261 Vserver Down Backup Hits                         --
        0
262
263 Current Multipath TCP sessions                   --
        0
264
265 Current Multipath TCP subflows                   --
        0
266
267 Apdex for client response times.                 --
       1.00
268
```



```
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:~#
<!--NeedCopy-->
```

NetScaler CPX のトラブルシューティング

November 23, 2023

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

- NetScaler CPX ログを表示する方法を教えてください。

  NetScaler CPX がオプションを使用して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。`tty:true` 次のコマンドを実行して、ログを表示できます:

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

  例:

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

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

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

<!--NeedCopy-->
  ```

  NetScaler CPX ファイルシステムの */cpx/log/boot.log* ファイルには、より多くのブートログがあります。

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

- NetScaler CPX からテクニカルサポートバンドルを入手する方法を教えてください。

  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すると、NetScaler CPX テクニカルサポートバンドルを収集できます。

```
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。`scpkubectl cp` またはを使用して、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します。

  例:

```
root@localhost# kubectl exec cpx-ingress1-55b9b6fc75-t5kc6 -c cpx
     -n citrix-adc /var/netscaler/bins/cli_script.sh "show
    techsupport"
exec: show techsupport
  Scope:  NODE
Done
root@localhost# kubectl cp cpx-ingress1-55b9b6fc75-t5kc6:var/tmp/
    support/collector_P_192.168.29.232_31Aug2020_07_30.tar.gz /tmp
    /collector_P_192.168.29.232_31Aug2020_07_30.tar.gz -c cpx
root@localhost# ll /tmp/collector_P_192.168.29.232
    _31Aug2020_07_30.tar.gz
-rw-r--r-- 1 root root 1648109 Aug 31 13:23 /tmp/collector_P_192
    .168.29.232_31Aug2020_07_30.tar.gz
```

- NetScaler CPX ポッドが起動中に動かなくなるのはなぜですか?

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

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

  例:

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

  ポッドイベントがコンテナが開始されたことを示している場合は、ポッドログを確認する必要があります。

- NetScaler CPX ポッドと Kubernetes マスターノード間でファイルをコピーする方法を教えてください。

  /`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には、Docker のボリュームマウント機能を使用することをお勧めします。NetScaler CPX コンテナがコアダンプを終了すると、ログやその他の重要なデータがマウントポイントで利用可能になります。

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

kubectl cp: 次のコマンドを実行して、ポッドからノードにファイルをコピーできます:

```
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: コマンドを使用して、NetScaler CPX ポッドと Kubernetes ノード間でファイルをコピーできます。次のコマンドを実行して、ポッドからノードにファイルをコピーします。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方法を教えてください。

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

  ```
  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
  ```

  次に、以下のコマンドを実行してパケットのキャプチャを開始します:

  ```
  cli_script.sh  “start nstrace -size 0”
  ```

  進行中のパケットキャプチャを停止する場合は、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/timestamp** ディレクトリにある**.cap** ファイルにキャプチャされたパケットを表示できます。

- NetScaler CPX を環境変数で展開しても、ライセンスサーバーが構成されないのはなぜですか? LS_IP=<ADM-IP>

  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ます。ping <ADM-IP> コマンドを使用して、NetScaler CPX ノードから NetScaler ADM への接続を確認できます。

  NetScaler ADM にノードからアクセスできる場合は、*/cpx/log/boot.log* ファイルの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イスで次のコマンドを使用して、ライセンスサーバーの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が正常に構成された後でも、NetScaler CPX でプールライセンスが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ます。NetScaler CPX ポッドのシェルインターフェイスで次のコマンドを使用して、NetScaler CPX で構成されたプール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 で正常に構成されたら、次のコマンドを使用してライセンスサーバーで使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イスで次のコマンドを実行します。

```
cli_script.sh  “sh licenseserverpool”
```

  例:


```
root@cpx-ingress-596d56bb6-zbx6h:/cpx/log# cli_script.sh "show
    licenseserverpool"
exec: show licenseserverpool
    Instance Total                    : 5
    Instance Available                : 4
    Standard Bandwidth Total          : 0 Mbps
    Standard Bandwidth Availabe       : 0 Mbps
    Enterprise Bandwidth Total        : 0 Mbps
    Enterprise Bandwidth Available    : 0 Mbps
    Platinum Bandwidth Total          : 10.00 Gbps
    Platinum Bandwidth Available      : 9.99 Gbps
    CP1000 Instance Total             : 100
    CP1000 Instance Available         : 100
  Done
<!--NeedCopy-->
```

- NITRO API 呼び出しに NetScaler *CPX* から接続拒否の応答が返されるのはなぜですか?

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

```
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
<!--NeedCopy-->
```

- NetScaler CPX の NSPPE プロセスが、トラフィックがない場合やほとんどない場合でも CPU 使用率の大部分を消費するのはなぜですか?

  NetScaler `CPX_CONFIG='{ "YIELD":"NO" } '` CPX を環境変数とともに展開すると、トラフィックがまったくない、またはほとんどない場合でも、NSPPE プロセスは CPU 使用率を 100% 消費します。NSPPE プロセスが CPU 使用率を消費しないようにするには、環境変数なしで NetScaler CPX を展開する必要があります。`CPX_CONFIG='{ "YIELD":"NO" }` 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 CPX が NetScaler ADM への登録に必要な環境変数とともに導入されたのに、NetScaler ADM に表示されないのはなぜですか?

  NetScaler CPX を NetScaler ADM に登録するためのログは、NetScaler CPX ファイルシステムの

/cpx/log/boot.log ファイルにあります。

コマンドを使用して、NetScaler CPX ポッドから NetScaler ADM IP アドレスにアクセスできるかどうかを確認できます。`ping` また、NetScaler ADM の登録に必要なすべての環境変数が NetScaler CPX コンテナ用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`: ノードの IP アドレスを指定します。
- `NS_HTTP_PORT`: ノードにマップされた HTTP ポートを指定します。
- `NS_HTTPS_PORT`: ノードにマップされた HTTPS ポートを指定します。
- `NS_SSH_PORT`: ノードにマップされた SSH ポートを指定します。
- `NS_SNMP_PORT`: ノードにマップされた 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「無効なユーザー名またはパスワードです」というエラーメッセージが表示されるのはなぜですか。

  `cli_script.sh` このコマンドは、NetScaler CPX 上の NSCLI のラッパーユーティリティです。最初の引数をコマンド文字列またはファイルパスとして実行し、2 番目の引数はオプションであり、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は、`cli_script.sh`の 2 番目の引数として資格情報を指定する必要があります。次のコマンドを実行して、資格情報を使用して NSCLI を実行できます:

```
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
```

- NetScaler CPX への SSH がユーザーに対して失敗するのはなぜですか? `rootnsroot`

13.0-64.35 リリース以降、NetScaler CPX は SSH ユーザー向けにデフォルトのパスワードを生成して更新します（および）。`rootnsroot` 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 `/var/deviceinfo/random_id` CPX のファイルシステムで確認できます。