

NSX-T Container Plug-in for Kubernetes - インストールおよび管理者ガイド

更新日：2017 年 9 月 7 日

VMware NSX-T Data Center 2.0



vmware®

最新の技術ドキュメントは、VMware の Web サイト (<https://docs.vmware.com/jp/>) でご確認いただけます。このドキュメントに関するご意見およびご感想は、docfeedback@vmware.com までお送りください。

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

ヴァイエルムウェア株式会社
105-0013 東京都港区浜松町 1-30-5
浜松町スクエア 13F
www.vmware.com/jp

Copyright © 2017 VMware, Inc. All rights reserved. [著作権および商標情報](#)。

目次

NSX-T Container Plug-in for Kubernetes - インストールおよび管理ガイド 4

1 NSX-T Container Plug-in の概要 5

互換性要件 5

インストールの概要 6

2 NSX-T リソースのセットアップ 7

NSX-T リソースの設定 7

Tier-0 論理ルーターの作成と設定 9

3 NCP と関連コンポーネントのインストール 11

NSX-T CNI プラグインのインストール 11

OVS のインストールと構成 12

Kubernetes ノードの NSX-T ネットワークの設定 13

NSX ノード エージェントのインストール 14

nsx-node-agent-ds.yml の ncp.ini の ConfigMap 15

NSX-T Container Plug-in のインストール 17

ncp-rc.yml の ncp.ini の ConfigMap 19

NCP ポッドでの PEM エンコードの証明書とプライベート キーのマウント 22

NCP ポッドでの証明書ファイルのマウント 22

Syslog の設定 23

Syslog の Sidecar コンテナの作成 23

Syslog の DaemonSet レプリカの作成 25

セキュリティの考慮事項 27

4 NSX-T Container Plug-in の管理 31

NSX Manager グラフィカル ユーザー インターフェイスでの IP ブロックの管理 31

NSX Manager グラフィカル ユーザー インターフェイスでの IP ブロックのサブネットの管理 32

CIF 接続論理ポート 32

CLI コマンド 33

NSX-T Container Plug-in for Kubernetes - インストールおよび管理ガイド

このガイドでは、NSX-T Container Plug-in (NCP) をインストールして管理する方法について説明します。また、コンテナベースのアプリケーション環境である Kubernetes と NSX-T の統合方法も説明します。

対象読者

このガイドは、システム管理者とネットワーク管理者を対象としています。ネットワークおよび仮想化テクノロジーに精通していることを前提としています。

VMware の技術ドキュメントの用語集

VMware は、新しい用語を集めた用語集を提供しています。当社の技術ドキュメントで使用されている用語の定義については、<http://www.vmware.com/support/pubs> をご覧ください。

NSX-T Container Plug-in の概要

1

NSX-T Container Plug-in (NCP) は、NSX-T と Kubernetes などのコンテナ オーケストレータとの統合や、NSX-T と OpenShift などのコンテナベースの PaaS (Platform as a Service) ソフトウェア製品との統合を可能にします。このガイドでは、NCP と Kubernetes のセットアップについて説明します。

NCP のメイン コンポーネントはコンテナで実行され、NSX Manager や Kubernetes 制御プレーンと通信を行います。NCP は、コンテナや他のリソースに対する変更を監視し、NSX API を呼び出して、コンテナの論理ポート、スイッチ、ルーター、セキュリティ グループなどのネットワーク リソースを管理します。

NSX CNI プラグインは、個々の Kubernetes ノードで実行されます。コンテナのライフ サイクル イベントを監視し、コンテナ インターフェイスをゲスト vSwitch に接続します。プログラムによってゲスト vSwitch をタグ付けし、コンテナ インターフェイスと vNIC 間でコンテナ トラフィックを転送します。

本リリースでは、NCP は単一の Kubernetes クラスタをサポートします。

この章には、次のトピックが含まれています。

- [互換性要件](#)
- [インストールの概要](#)

互換性要件

NSX-T Container Plug-in には、Kubernetes および Ubuntu に関連する次の互換性要件があります。

ソフトウェア製品	バージョン
コンテナ ホストの仮想マシンのハイパーバイザー	<ul style="list-style-type: none">■ ESXi 6.5■ ESXi 6.5 Update 1■ RHEL KVM 7.3■ Ubuntu KVM 16.04
コンテナ ホストのオペレーティング システム	<ul style="list-style-type: none">■ RHEL 7.3■ Ubuntu 16.04
コンテナ ランタイム	Docker 1.12
Container Orchestrator	Kubernetes 1.5、1.6
ゲスト vSwitch	OVS 2.6、2.7

インストールの概要

Kubernetes がインストールされている環境では、通常、次の手順で NCP のインストールと構成を行います。

- 1 NSX-T をインストールします。
- 2 オーバーレイ トランспорт ゾーンを作成します。
- 3 オーバーレイ 論理スイッチを作成し、Kubernetes ノードをスイッチに接続します。
- 4 Tier-0 論理ルーターを作成します。
- 5 Kubernetes ポッドの IP アドレス ブロックを作成します。
- 6 SNAT（送信元ネットワーク アドレス変換）用の IP アドレス ブロックまたは IP アドレス プールを作成します。
- 7 各ノードに NSX CNI（コンテナ ネットワーク インターフェイス）プラグインをインストールします。
- 8 各ノードに OVS (Open vSwitch) をインストールします。
- 9 Kubernetes ノードに NSX-T ネットワークを構成します。
- 10 DaemonSet として NSX ノード エージェントをインストールします。
- 11 ReplicationController として NCP をインストールします。
- 12 NCP ポッドにセキュリティ証明書をマウントします。

NSX-T リソースのセットアップ

2

NSX-T Container Plug-in をインストールする前に、特定の NSX-T リソースを設定する必要があります。

この章には、次のトピックが含まれています。

- [NSX-T リソースの設定](#)
- [Tier-0 論理ルーターの作成と設定](#)

NSX-T リソースの設定

設定が必要な NSX-T リソースには、オーバーレイ トランスポート ゾーン、Tier-0 論理ルーター、ノード仮想マシンに接続する論理スイッチ、Kubernetes ノードの IP アドレス ブロック、SNAT の IP アドレス ブロックまたはプールなどがあります。

オーバーレイ トランスポート ゾーン

クラスタのオーバーレイ トランスポート ゾーンは、{'ncp/cluster': '<cluster_name>'} タグで識別されます。NSX Manager にログインし、[ファブリック (Fabric)] - [トランスポート ゾーン (Transport Zones)] の順に移動します。ネットワーク コンテナに使用されているオーバーレイ トランスポート ゾーンを検索するか、新規に作成します。設定しているクラスタの名前でトランスポート ゾーンにタグを付けます。<cluster_name> は、ncp.ini の [coe] セクションにある cluster オプションの値と一致させる必要があります。共有するトランスポート ゾーンには複数のタグを追加できます。

Tier-0 論理ルーティング

クラスタの Tier-0 論理ルーターは {'ncp/cluster': '<cluster_name>'} タグで識別されます。NSX Manager にログインし、[ルーティング (Routing)] - [ルーター (ROUTERS)] の順に移動します。Kubernetes クラスタに新しい Tier-0 論理ルーターを作成するか、既存のルーターを使用します。ルーターを特定したら、そのルーターに {'ncp/cluster': '<cluster_name>'} タグを付けます。

<cluster_name> 値は、ncp.ini の [coe] セクションにあるクラスタ オプションの値と一致させる必要があります。共有するルーターには複数のタグを追加できます。

注： ルーターは、アクティブ/スタンバイ モードで作成する必要があります。

論理スイッチ

ネットワーク トラフィック用のポッドが使用するインターフェイスは、オーバーレイ論理スイッチに接続する必要があります。ノードの管理インターフェイスは NSX-T に接続する必要はありませんが、接続すると設定が簡単になります。スイッチは、`{'ncp/cluster': '<cluster_name>'}` タグで識別されます。NSX Manager にログインし、[スイッチング (Switching)] - [スイッチ (Switches)] の順に移動します。新しいスイッチを作成するか、既存のスイッチを使用します。スイッチを特定したら、そのスイッチに `{'ncp/cluster': '<cluster_name>'}` タグを付けます。

`<cluster_name>` 値は、`ncp.ini` の `[coe]` セクションにあるクラスタ オプションの値と一致させる必要があります。共有するスイッチには複数のタグを追加できます。

ノードの仮想マシンをスイッチに接続します。手順については、NSX-T 管理ガイドの「論理スイッチへの仮想マシンの接続」を参照してください。

Kubernetes ポッドの IP アドレス ブロック

Kubernetes ポッドに 1 つ以上の IP アドレス ブロックを作成します。NSX Manager にログインして、[DDI] - [IP アドレス管理 (IPAM)] の順に移動し、IP アドレス ブロックを作成します。IP アドレス ブロックを CIDR 形式で指定します。ブロックに `ncp/cluster` タグを指定します。

SNAT 以外の名前空間に IP アドレス ブロックを作成することもできます。これらの IP アドレス ブロックには、`ncp/cluster` タグだけでなく、`{'ncp/no_snat': '<cluster_name>'}` タグも必要です。NCP の実行中に SNAT 以外の IP アドレス ブロックを作成する場合には、NCP を再起動する必要があります。再起動しないと、IP アドレス ブロックが不足するまで、NCP は共有の IP アドレス ブロックを使用します。

注： IP アドレス ブロックを作成するときに、NCP 構成ファイル `ncp.ini` の `subnet_prefix` パラメータの値よりも大きいプリフィックスは使用しないでください。詳細については、[ncp-rc.yml](#) の `ncp.ini` の `ConfigMap` を参照してください。

SNAT の IP アドレス ブロックまたは IP アドレス プール

これらのリソースは、SNAT ルールによるポッドの IP アドレス変換用の IP アドレスの割り当てに使用されます。この IP アドレスは、OpenStack フローティング IP アドレスなど、SNAT/DNAT ルールで入力側のコントローラを公開する場合にも使用されます。このガイドでは、これらの IP アドレスを外部 IP アドレスともいいます。ユーザーは、グローバル外部 IP アドレス ブロックまたはクラスタ固有の外部 IP アドレス プールを設定できます。

外部の IP アドレス ブロックを設定するには、NSX Manager にログインして、[DDI] - [IP アドレス管理 (IPAM)] の順に移動します。ホストのアドレスではなくネットワークアドレスを使用して CIDR 値を指定します。たとえば、`4.3.2.1/16` ではなく `4.3.0.0/16` を指定します。次のキーと値を使用して、外部 IP アドレス用に IP アドレス ブロックにタグを付けます。

```
{'ncp/external': 'true'}
```

複数の Kubernetes クラスタが同じ外部 IP アドレス プールを使用します。各 NCP インスタンスは、管理する Kubernetes クラスタにこのプールのサブセットを使用します。デフォルトでは、ポッドのサブネットに同じサブネットのプリフィックスが使用されます。異なるサイズのサブネットを使用するには、`ncp.ini` の `[nsx_v3]` セクションにある `external_subnet_prefix` オプションを更新します。

外部 IP アドレスの割り当てにクラスタ固有の IP アドレス プールを使用するには、NSX Manager にログインして、[インベントリ (Inventory)] - [グループ (Groups)] - [IP アドレス プール (IP POOL)] の順に移動します。プールを作成するか、既存のプールを使用します。次のタグをプールに適用します。

```
{'ncp/cluster': 'true'}
{'ncp/external': 'true'}
```

(オプション) ファイアウォールのマーカー セクション

管理者が、NCP が作成したファイアウォール セクションと矛盾しないファイアウォール ルールを作成するには、NSX Manager にログインして、[ファイアウォール (Firewall)] - [全般 (General)] の順に移動し、空のファイアウォール セクションを作成して {'ncp/fw_sect_marker': 'true'} タグを付けます。このマーカー ファイアウォール セクションを作成すると、NCP がネットワーク ポリシー用に作成する後続のファイアウォール セクションと名前空間の隔離は、このファイアウォール セクションの上に配置されます。管理者が作成したファイアウォール ルールは、このマーカー ファイアウォール セクションの下に配置されます。

このマーカー セクションが作成されていない場合、隔離ルールはすべて一番下に作成されます。クラスタごとに複数のマーカー ファイアウォール セクションを作成することはできません。作成すると、エラーが発生します。

Tier-0 論理ルーターの作成と設定

Tier-0 論理ルーターは、Kubernetes ノードを介して外部ネットワークに接続します。

手順

- 1 ブラウザから、NSX Manager (<https://nsx-manager-ip-address>) にログインします。
- 2 [ルーティング (Routing)] - [ルーター (Routers)] の順に移動して、[追加 (Add)] - [Tier-0 ルーター (Tier-0 Router)] の順にクリックします。
- 3 名前を入力します。必要に応じて説明も入力します。
- 4 この Tier-0 論理ルーターをバックアップする既存の Edge クラスタをドロップダウン メニューから選択します。
- 5 (必須) 高可用性モードを選択します。
アクティブ/スタンバイを選択します。
- 6 [保存 (Save)] をクリックします。
新しい論理ルーターがリンクとして表示されます。
- 7 (必須) 論理ルーターのリンクをクリックします。
- 8 [ルーティング (Routing)] - [ルート再配分 (Route Redistribution)] の順にクリックします。
- 9 新しい再配分の基準を追加するには、[追加 (Add)] をクリックします。
送信元の場合には、ルート化された (NAT 以外の) トポロジで [NSX スタティック (NSX Static)] を選択します。
NAT トポロジの場合には、[Tier-0 NAT] を選択します。
- 10 [保存 (Save)] をクリックします。

- 11 新しく作成されたルーターをクリックします。
- 12 [設定 (Configuration)] - [ルーター ポート (Router Ports)] の順にクリックします。
- 13 [追加 (Add)] をクリックして、アップリンク ポートを追加します。
- 14 トランスポート ノードを選択します。
- 15 以前作成した論理スイッチを選択します。
- 16 外部ネットワークの IP アドレスを指定します。
- 17 [保存 (Save)] をクリックします。

新しい論理ルーターがリンクとして表示されます。

NCP と関連コンポーネントのインストール

3

NSX-T Container Plug-in (NCP) をインストールするには、マスター ノードと Kubernetes ノードにコンポーネントをインストールする必要があります。

この章には、次のトピックが含まれています。

- [NSX-T CNI プラグインのインストール](#)
- [OVS のインストールと構成](#)
- [Kubernetes ノードの NSX-T ネットワークの設定](#)
- [NSX ノード エージェントのインストール](#)
- [nsx-node-agent-ds.yml の ncp.ini の ConfigMap](#)
- [NSX-T Container Plug-in のインストール](#)
- [ncp-rc.yml の ncp.ini の ConfigMap](#)
- [NCP ポッドでの PEM エンコードの証明書とプライベート キーのマウント](#)
- [NCP ポッドでの証明書ファイルのマウント](#)
- [Syslog の設定](#)
- [セキュリティの考慮事項](#)

NSX-T CNI プラグインのインストール

Kubernetes ノードに NSX-T CNI プラグインをインストールします。

手順

- 1 Linux ディストリビューション用の、適切なインストール ファイルをダウンロードします。

ファイル名は `nsx-cni-1.0.0.0.0.xxxxxxx-1.x86_64.rpm` または `nsx-cni-1.0.0.0.0.xxxxxxx.deb` です。xxxxxxx はビルド番号です。

- 手順 1 でダウンロードした rpm または deb ファイルをインストールします。

プラグインが `/opt/cni/bin` にインストールされます。CNI 構成ファイル `10.net.conf` が `/etc/cni/net.d` にコピーされます。rpm は、ループバック プラグインの構成ファイル `/etc/cni/net.d/99-loopback.conf` もインストールします。

OVS のインストールと構成

ミニオン ノードに OVS (Open vSwitch) をインストールし、構成します。

手順

- Linux ディストリビューションのインストール ファイルをダウンロードします。

ファイル名は `openvswitch-common_2.7.0.6383646-1_amd64.deb`、`openvswitch-datapath-dkms_2.7.0.xxxxxxx-1_all.deb`、`openvswitch-switch_2.7.0.xxxxxxx-1_amd64.deb` です。
`xxxxxxx` はビルド番号です。

- 手順 1 でダウンロードした rpm または deb ファイルをインストールします。
- Ubuntu の場合、次のコマンドを実行して、OVS カーネル モジュールを再ロードします。

```
service openvswitch-switch force-reload-kmod
```

- OVS が実行されていることを確認します。

```
# service openvswitch-switch status
```

- まだ作成されていない場合には、`br-int` インスタンスを作成します。

```
# ovs-vsctl add-br br-int
```

- ノードの論理スイッチに接続するネットワーク インターフェイス (`node-if`) を `br-int` に追加します。

```
# ovs-vsctl add-port br-int <node-if> -- set Interface <node-if> ofport_request=1
```

次のコマンドを実行して、`ofport` を確認します。`ofport 1` が使用できない場合、OVS が使用可能なポートを割り当てるためです。

```
# ovs-vsctl --columns=ofport list interface <node-if>
```

`ofport` が 1 でない場合、NSX Node Agent の DaemonSet yaml ファイルの `nsx_kube_proxy` セクションに、`ovs_uplink_port` オプションを設定します。

- `br-int` と `node-iflink` が起動していることを確認します。

```
# ip link set br-int up
# ip link set <node-if> up
```

8 再起動後にネットワーク インターフェイスが起動するように、ネットワーク構成ファイルを更新します。

Ubuntu の場合、`/etc/network/interfaces` を更新して、次の行を追加します。

```
auto <node-if>
iface <node-if> inet manual
up ip link set <node-if> up
```

RHEL の場合、`/etc/sysconfig/network-scripts/ifcfg-<node-if>` を更新して、次の行を追加します。

```
ONBOOT=yes
```

Kubernetes ノードの NSX-T ネットワークの設定

このセクションでは、Kubernetes マスター ノードとワーカー ノードに NSX-T ネットワークを設定する方法について説明します。

各ノードには、2 つ以上のネットワーク インターフェイスが必要です。最初のインターフェイスは管理インターフェイスです。NSX-T ファブリック上になくてもかまいません。もう 1 つのインターフェイスはポッドにネットワークを提供します。このインターフェイスは NSX-T ファブリック上にあり、ノードの論理スイッチとして参照される論理スイッチに接続します。Kubernetes の健全性チェックが正常に機能するには、管理インターフェイスとポッドの IP アドレスがルーティング可能でなければなりません。管理インターフェイスとポッド間の通信では、NCP が健全性チェックと他の管理トラフィックを許可する分散ファイアウォール ルールを自動的に作成します。このルールの詳細は、NSX Manager のグラフィカル ユーザー インターフェイスで確認できます。このルールは、変更したり、削除しないでください。

各ノードの仮想マシンで、コンテナ ネットワーク用の vNIC がノード論理スイッチに接続していることを確認します。

各ノードのコンテナ トラフィックに使用されている vNIC の VIF ID は、NSX-T Container Plug-in (NCP) に認識されている必要があります。対応する論理スイッチ ポートに、次のようにタグを付ける必要があります。

```
{'ncp/node_name': '<node_name>'}
{'ncp/cluster': '<cluster_name>'}
```

NSX Manager のグラフィカル ユーザー インターフェイスから [インベントリ] - [仮想マシン] の順に移動すると、ノード仮想マシンの論理スイッチ ポートを確認できます。

Kubernetes ノード名が変更された場合には、`ncp/node_name` タグを更新し、NCP を再起動する必要があります。ノード名を取得するには、次のコマンドを使用します。

```
kubect1 get nodes
```

NCP の実行中にクラスタにノードを追加するには、`kubeadm join` コマンドを実行する前に論理スイッチ ポートにタグを追加する必要があります。この操作を行わないと、新しいノードはネットワークに接続できません。タグが間違っているか、見つからない場合、次の手順で問題を解決してください。

- 論理スイッチ ポートに正しいタグを適用します。

- NCP を再起動します。

NSX ノード エージェントのインストール

NSX ノード エージェントは、各ポッドで 2 つのコンテナを実行する DaemonSet です。1 つのコンテナは、NSX ノード エージェントを実行し、主にコンテナのネットワーク インターフェイスを管理します。そして、CNI プラグインや Kubernetes API サーバと通信を行います。もう 1 つのコンテナは、NSX kube-proxy を実行し、クラスタの IP アドレスをポッドの IP アドレスに変換することで Kubernetes サービスの抽象化を実装します。これは、アップストリームの kube-proxy と同じ機能です。

手順

- 1 NCP Docker イメージをダウンロードします。

ファイル名は `nsx-ncp-xxxxxxx.tar` です。xxxxxxx はビルド番号です。

- 2 NSX ノード エージェントの DaemonSet yaml テンプレートをダウンロードします。

ファイル名は `ncp-node-agent-ds.yaml` です。このファイルを編集することも、独自のテンプレート ファイルの例として使用することもできます。

- 3 NCP Docker イメージをイメージ レジストリにロードします。

```
docker load -i <tar file>
```

- 4 `ncp-node-agent-ds.yaml` を編集します。

ロードされたイメージの名前を変更します。

Ubuntu の場合、yaml ファイルは AppArmor が有効になっていることを前提とします。AppArmor が有効かどうかは、`/sys/module/apparmor/parameters/enabled` ファイルで確認します。AppArmor が有効でない場合には、次のように変更します。

- 次の行を削除するか、コメント行にします。

```
container.apparmor.security.beta.kubernetes.io/nsx-node-agent: localhost/node-agent-  
apparmor
```

nsx-node-agent コンテナと nsx-kube-proxy コンテナの securityContext の下に行 privileged:true を追加します。次はその例です。

```
securityContext:
  privileged:true
```

注： hyperkube イメージを使用するコンテナ内で kubelet を実行すると、実際の状態に関係なく、kubelet が AppArmor の状態を常に無効とレポートする問題が確認されています。yaml ファイルでも、同様の変更を行う必要があります。

注： yaml ファイルで、ncp.ini に生成された ConfigMap を読み取り専用ボリュームとしてマウントするように指定する必要があります。ダウンロードした yaml ファイルでは、すでにこれが指定されています。この設定は変更できません。

- 5 次のコマンドを使用して、NSX ノード エージェントの DaemonSet を作成します。

```
kubectl apply -f ncp-node-agent-ds.yml
```

nsx-node-agent-ds.yml の ncp.ini の ConfigMap

サンプルの YAML ファイル nsx-node-agent-ds.yml には、NSX Node Agent の構成ファイル ncp.ini の ConfigMap が含まれています。この ConfigMap セクションには、Node Agent のインストールをカスタマイズするためのパラメータが含まれています。

ダウンロードしたサンプルの nsx-node-agent-ds.yml には、次の ncp.ini 情報が含まれます。

```
# ConfigMap for ncp.ini
apiVersion: v1
kind: ConfigMap
metadata:
  name: nsx-node-agent-config
  labels:
    version: v1
data:
  ncp.ini: |
    [DEFAULT]

    # Set to True to enable logging to stderr
    use_stderr = True
    # Set to True to send logs to the syslog daemon
    # use_syslog = False
    # Enabler debug-level logging for the root logger. If set to True, the
    # root logger debug level will be DEBUG, otherwise it will be INFO.
    # debug = True

    # Log file path for NCP operations.
    log_dir = /var/log/nsx-ujio/

    [coe]
    #
```

```

# Common options for Container Orchestrators
#

# Container orchestrator adaptor to plug in
# Options: kubernetes (default), cloud-foundry, openshift
# adaptor = kubernetes

# Specify cluster for adaptor. It is a prefix of NSX resources name to
# distinguish multiple clusters who are using the same NSX.
# This is also used as the tag of IP blocks for cluster to allocate
# IP addresses. Different clusters should have different IP blocks.
# cluster = k8scluster

# Log level for the NCP operations. If set, overrides the level specified
# for the root logger. Possible values are NOTSET, DEBUG, INFO, WARNING,
# ERROR, CRITICAL
#loglevel=None

# Log level for the NSX API client operations. If set, overrides the level
# specified for the root logger. Possible values are NOTSET, DEBUG, INFO,
# WARNING, ERROR, CRITICAL
nsxlib_loglevel=INFO

[k8s]
#
# From kubernetes
#

# IP address of the Kubernetes API Server. If not set, will try to
# read and use the Kubernetes Service IP from environment variable
# KUBERNETES_SERVICE_HOST.
#apiserver_host_ip = <ip_address>

# Port of the Kubernetes API Server.
# Set to 6443 for https. If not set, will try to
# read and use the Kubernetes Service port from environment
# variable KUBERNETES_SERVICE_PORT.
#apiserver_host_port = <port>

# Specify a CA bundle file to use in verifying the Kubernetes API server
# certificate. (string value)
#ca_file = <None>
ca_file = /var/run/secrets/kubernetes.io/serviceaccount/ca.crt

# Full path of the Token file to use for authenticating with the k8s API server.
#client_token_file = <None>
client_token_file = /var/run/secrets/kubernetes.io/serviceaccount/token

# Full path of the client certificate file to use for authenticating
# with the k8s API server. It must be specified together with
# "client_private_key_file"
#client_cert_file = <None>

# Full path of the client certificate file to use for authenticating
# with the k8s API server. It must be specified together with

```



```
# "client_cert_file"
#client_private_key_file = <None>

# Log level for the kubernetes adaptor. If set, overrides the level specified
# for the root logger. Possible values are NOTSET, DEBUG, INFO, WARNING,
# ERROR, CRITICAL
#loglevel=None

[nsx_node_agent]
#
# Configuration for nsx_node_agent
#

# Needs to mount node /proc to container if nsx_node_agent runs in a container.
# By default node /proc will be mounted to /host/proc, the prefix is /host.
# It should be the same setting with mounted path in the daemonset yaml file.
# Set the path to '' if nsx_node_agent is running as a process in minion node.
#proc_mount_path_prefix = /host

# The IP address for nsx_node_agent to communicate with NSX RPC server.
# The format should be ip/mask.
#nsxrpc_cip = 169.254.1.0/31

# The port for nsx_node_agent to communicate with NSX RPC server.
#nsxrpc_port = 2345

# The vlan id for nsx_node_agent to communicate with NSX RPC server.
#nsxrpc_vlan = 4094

# The interval of NSX RPC keep alive message.
#nsxrpc_keepalive_interval = 3

[nsx_kube_proxy]
#
# Configuration for nsx_kube_proxy
#

# The OVS uplink OpenFlow port where to apply the NAT rules to.
# If not specified, the port that gets assigned ofport=1 is used.
#ovs_uplink_port = <None>
```

NSX-T Container Plug-in のインストール

NSX-T Container Plug-in (NCP) は、Docker イメージとして配信されます。NCP は、インフラストラクチャ サービスのノードで実行する必要があります。マスター ノードでの NCP の実行は推奨されません。

手順

- 1 NCP Docker イメージをダウンロードします。

ファイル名は `nsx-ncp-xxxxxxx.tar` です。xxxxxxx はビルド番号です。

2 NCP ReplicationController yaml テンプレートをダウンロードします。

ファイル名は `ncp-rc.yaml` です。このファイルを編集することも、独自のテンプレート ファイルの例として使用することもできます。

3 NCP Docker イメージをイメージ レジストリにロードします。

```
docker load -i <tar file>
```

4 `ncp-rc.yaml` を編集します。

ロードされているイメージの名前を変更します。

`nsx_api_managers` パラメータを指定します。このリリースでは、1 つの Kubernetes ノード クラスと 1 つの NSX Manager インスタンスがサポートされます。次はその例です。

```
nsx_api_managers = 192.168.1.180
```

(オプション) `[nsx_v3]` セクションで `ca_file` パラメータを指定します。値は、NSX Manager サーバ証明書の検証に使用する CA バンドル ファイルにする必要があります。設定しないと、システムのルート CA が使用されます。

NSX-T で認証を行うため、`nsx_api_cert_file` と `nsx_api_private_key_file` パラメータを指定します。

`nsx_api_cert_file` は、PEM 形式のクライアント証明書ファイルのフル パスです。このファイルの内容は次のようになります。

```
-----BEGIN CERTIFICATE-----
<certificate_data_base64_encoded>
-----END CERTIFICATE-----
```

`nsx_api_private_key_file` は、PEM 形式のクライアント プライベート キー ファイルのフル パスです。このファイルの内容は次のようになります。

```
-----BEGIN PRIVATE KEY-----
<private_key_data_base64_encoded>
-----END PRIVATE KEY-----
```

入力側のコントローラを NAT モードで実行するように構成する場合には、`ingress_mode = nat` パラメータを指定します。

デフォルトでは、ポッドの論理スイッチで IP アドレス ブロックから割り当てられたすべてのサブネットにサブネット プリフィックス 24 が使用されます。別のサブネット サイズを使用するには、`[nsx_v3]` セクションの `subnet_prefix` オプションを更新します。

注： yaml ファイルで、`ncp.ini` に生成された ConfigMap を読み取り専用ボリュームとしてマウントするように指定する必要があります。ダウンロードした yaml ファイルには、この指定がすでに行われています。この設定は変更できません。

5 NCP ReplicationController を作成します。

```
kubectl create -f ncp-rc.yml
```

ncp-rc.yml の ncp.ini の ConfigMap

サンプルの YAML ファイル `ncp-rc.yml` には、構成ファイル `ncp.ini` の ConfigMap が含まれています。この ConfigMap セクションには、前のセクションで説明したように、NCP のインストール前に指定する必要があるパラメータが含まれます。

ダウンロードしたサンプルの `ncp-rc.yml` には、次の `ncp.ini` 情報が含まれます。

```
# ConfigMap for ncp.ini
apiVersion: v1
kind: ConfigMap
metadata:
  name: nsx-ncp-config
  labels:
    version: v1
data:
  ncp.ini: |
    [DEFAULT]

    # Set to True to enable logging to stderr
    use_stderr = True
    # Set to True to send logs to the syslog daemon
    # use_syslog = False
    # Enabler debug-level logging for the root logger. If set to True, the
    # root logger debug level will be DEBUG, otherwise it will be INFO.
    # debug = True

    # Log file path for NCP operations.
    log_dir = /var/log/nsx-ujo/

    [coe]
    #
    # Common options for Container Orchestrators
    #

    # Container orchestrator adaptor to plug in
    # Options: kubernetes (default), cloud-foundry, openshift
    # adaptor = kubernetes

    # Specify cluster for adaptor. It is a prefix of NSX resources name to
    # distinguish multiple clusters who are using the same NSX.
    # This is also used as the tag of IP blocks for cluster to allocate
    # IP addresses. Different clusters should have different IP blocks.
    # Each cluster in an NSX installation must have a unique name.
    # cluster = k8scluster
```

```

# Log level for the NCP operations. If set, overrides the level specified
# for the root logger. Possible values are NOTSET, DEBUG, INFO, WARNING,
# ERROR, CRITICAL
#loglevel=None

# Log level for the NSX API client operations. If set, overrides the level
# specified for the root logger. Possible values are NOTSET, DEBUG, INFO,
# WARNING, ERROR, CRITICAL
nsxlib_loglevel=INFO

[k8s]
#
# From kubernetes
#

# IP address of the Kubernetes API Server. If not set, will try to
# read and use the Kubernetes Service IP from environment variable
# KUBERNETES_SERVICE_HOST.
#apiserver_host_ip = <ip_address>

# Port of the Kubernetes API Server.
# Set to 6443 for https. If not set, will try to
# read and use the Kubernetes Service port from environment
# variable KUBERNETES_SERVICE_PORT.
#apiserver_host_port = <port>

# Specify a CA bundle file to use in verifying the Kubernetes API server
# certificate. (string value)
#ca_file = <None>
ca_file = /var/run/secrets/kubernetes.io/serviceaccount/ca.crt

# Full path of the Token file to use for authenticating with the k8s API server.
#client_token_file = <None>
client_token_file = /var/run/secrets/kubernetes.io/serviceaccount/token

# Full path of the client certificate file to use for authenticating
# with the k8s API server. It must be specified together with
# "client_private_key_file"
#client_cert_file = <None>

# Full path of the client certificate file to use for authenticating
# with the k8s API server. It must be specified together with
# "client_cert_file"
#client_private_key_file = <None>

# Log level for the kubernetes adaptor. If set, overrides the level specified
# for the root logger. Possible values are NOTSET, DEBUG, INFO, WARNING,
# ERROR, CRITICAL
#loglevel=None

# Specify how ingress controllers are expected to be deployed. Possible values:
# hostnetwork or nat. NSX will create NAT rules only in the second case.
#ingress_mode = hostnetwork

[nsx_v3]

```

```

#
# From nsx
#

# IP address of one or more NSX managers separated by commas. The IP address
# should be of the form (list value):
# <ip_address1>[:<port1>],<ip_address2>[:<port2>],...
# HTTPS will be used for communication with NSX. If port is not provided,
# port 443 will be used.
#nsx_api_managers = <ip_address>

# Specify a CA bundle file to use in verifying the NSX Manager server
# certificate. This option is ignored if "insecure" is set to True. If
# "insecure" is set to False and ca_file is unset, the system root CAs will be
# used to verify the server certificate. (string value)
#ca_file = <None>

# Path to NSX client certificate file. If specified, the nsx_api_user and
# nsx_api_password options will be ignored. This option must be specified
# along with "nsx_api_private_key_file" option.
# nsx_api_cert_file = <None>

# Path to NSX client private key file. If specified, the nsx_api_user and
# nsx_api_password options will be ignored. This option must be specified
# along with "nsx_api_cert_file" option.
# nsx_api_private_key_file = <None>

# The time in seconds before aborting a HTTP connection to a NSX manager.
# (integer value)
#http_timeout = 10

# The time in seconds before aborting a HTTP read response from a NSX manager.
# (integer value)
#http_read_timeout = 180

# Maximum number of times to retry a HTTP connection. (integer value)
#http_retries = 3

# Maximum concurrent connections to each NSX manager. (integer value)
#concurrent_connections = 10

# The amount of time in seconds to wait before ensuring connectivity to the NSX
# manager if no manager connection has been used. (integer value)
#conn_idle_timeout = 10

# Number of times a HTTP redirect should be followed. (integer value)
#redirects = 2

# Maximum number of times to retry API requests upon stale revision errors.
# (integer value)
#retries = 10

# Subnet prefix of IP block. IP block will be retrieved from NSX API and
# recognised by tag 'cluster'.
# Prefix should be less than 31, as two addresses(the first and last addresses)

```

```
# need to be network address and broadcast address.
# The prefix is fixed after the first subnet is created. It can be changed only
# if there is no subnets in IP block.
#subnet_prefix = 24

# Subnet prefix of external IP block. Use subnet_prefix if not specified.
#external_subnet_prefix = <None>

# Indicates whether distributed firewall DENY rules are logged.
#log_dropped_traffic = False
```

NCP ポッドでの PEM エンコードの証明書とプライベート キーのマウント

PEM でエンコードされた証明書とプライベート キーがある場合には、yaml ファイルで NCP ポッド定義を更新して、NCP ポッドに TLS Secret をマウントできます。

- 1 証明書とプライベート キーに TLS Secret を作成します。

```
kubectl create secret tls SECRET_NAME --cert=/path/to/tls.crt --key=/path/to/tls.key
```

- 2 NCP ポッド仕様 yaml を更新して、NCP ポッド仕様にファイルとして Secret をマウントします。

```
spec:
  ...
  containers:
  - name: nsx-ncp
    ...
    volumeMounts:
    ...
    - name: nsx-cert
      mountPath: /etc/nsx-uj0/nsx-cert
      readOnly: true
  volumes:
  ...
  - name: nsx-cert
    secret:
      secretName: SECRET_NAME
```

- 3 yaml ファイルで nsx_v3 オプションの nsx_api_cert_file と nsx_api_private_key_file を更新します。

```
nsx_api_cert_file = /etc/nsx-uj0/nsx-cert/tls.crt
nsx_api_private_key_file = /etc/nsx-uj0/nsx-cert/tls.key
```

NCP ポッドでの証明書ファイルのマウント

ノードのファイル システムに証明書ファイルがある場合、NCP ポッド仕様を更新して NCP ポッドにファイルをマウントできます。

次はその例です。

```
spec:
  ...
  containers:
  - name: nsx-ncp
    ...
    volumeMounts:
    ...
    - name: nsx-cert
      # Mount path must match nsx_v3 option "nsx_api_cert_file"
      mountPath: /etc/nsx-uj0/nsx_cert
    - name: nsx-priv-key
      # Mount path must match nsx_v3 option "nsx_api_private_key_file"
      mountPath: /etc/nsx-uj0/nsx_priv_key
  volumes:
  ...
  - name: nsx-cert
    hostPath:
      path: <host-filesystem-cert-path>
  - name: nsx-priv-key
    hostPath:
      path: <host-filesystem-priv-key-path>
```

Syslog の設定

コンテナで rsyslog や syslog-ng などの Syslog エージェントを実行すると、NCP や関連コンポーネントのログを Syslog サーバに送信できます。

次の方法を推奨します。Kubernetes でのログの詳細については、<https://kubernetes.io/docs/concepts/cluster-administration/logging> を参照してください。

- NCP または nsx-node-agent ポッドで実行する Sidecar コンテナを作成します。
- すべてのノードで DaemonSet レプリカを実行します。

注： プラグインはポッドで実行されないため、Sidecar コンテナでは NSX CNI プラグイン ログを Syslog サーバに送信できません。

Syslog の Sidecar コンテナの作成

NCP と同じポッドで Syslog が実行されるように、Sidecar コンテナを設定できます。次の手順では、Syslog エージェント イメージが example/rsyslog であることを前提としています。

手順

- 1 ログをファイルに書き込むように、NCP と NSX のノード エージェントを設定します。

NCP と NSX ノード エージェントの yaml ファイルで、log_dir パラメータを設定し、マウントするボリュームを指定します。次に例を示します。

```
[default]
log_dir = /var/log/nsx-ujo/
...

spec:
  ...
  containers:
    - name: nsx-ncp
      ...
      volumeMounts:
        - name: nsx-ujo-log-dir
          # Mount path must match [default] option "log_dir"
          mountPath: /var/log/nsx-ujo
  volumes:
    ...
    - name: nsx-ujo-log-dir
      hostPath:
        path: <host-filesystem-log-dir-path>
```

ログ ファイルの名前を変更するには、log_file パラメータを設定します。デフォルトの名前は、ncp.log、nsx_node_agent.log、nsx_kube_proxy.log です。log_dir オプションに /var/log/nsx-ujo 以外のパスが設定されている場合、hostPath ボリュームまたは emptyDir ボリュームを作成して、対応するポッド仕様にマウントする必要があります。

- 2 NCP ポッド仕様 yaml ファイルに、Syslog の ConfigMap を追加します。次に例を示します。

```
kind: ConfigMap
metadata:
  name: rsyslog-config
  labels:
    version: v1
data:
  ncp.conf: |
    module(load="imfile")

    ruleset(name="remote") {
      action(type="omfwd"
        Protocol="tcp"
        Target="nsx.example.com"
        Port="514")

      stop
    }
}
```



```
input(type="imfile"
      File="/var/log/nsx-ujo/ncp.log"
      Tag="ncp"
      Ruleset="remote"
```

- 3 NCP ポッドの yaml ファイルに rsyslog コンテナに追加します。rsyslog が構成データを検索し、他のコンテナからログを読み取れるボリュームをマウントします。次に例を示します。

```
spec:
  containers:
  - name: nsx-ncp
    ...
  - name: rsyslog
    image: example/rsyslog
    imagePullPolicy: IfNotPresent
    volumeMounts:
    - name: rsyslog-config-volume
      mountPath: /etc/rsyslog.d
      readOnly: true
    - name: nsx-ujo-log-dir
      mountPath: /var/log/nsx-ujo
  volumes:
  ...
  - name: rsyslog-config-volume
    configMap:
      name: rsyslog-config
  - name: nsx-ujo-log-dir
    hostPath:
      path: <host-filesystem-log-dir-path>
```

Syslog の DaemonSet レプリカの作成

この方法では、NCP のすべてのコンポーネントのログをリダイレクトできます。アプリケーションは、デフォルトで有効になっている stderr にログを記録するように設定する必要があります。次の手順では、Syslog エージェントイメージが example/rsyslog であることを前提としています。

手順

- 1 DaemonSet yaml ファイルを作成します。次に例を示します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: rsyslog-config
  labels:
    version: v1
data:
  nsx-ncp.conf: |
    module(load="imfile")

    ruleset(name="remote") {
      if $msg contains 'nsx-container' then
        action(type="omfwd"
```

```

        Protocol="tcp"
        Target="nsx.example.com"
        Port="514")

    stop
}

input(type="imfile"
      File="/var/log/containers/nsx-node-agent-*.log"
      Tag="nsx-node-agent"
      Ruleset="remote")

input(type="imfile"
      File="/var/log/containers/nsx-ncp-*.log"
      Tag="nsx-ncp"
      Ruleset="remote")

input(type="imfile"
      File="/var/log/syslog"
      Tag="nsx-cni"
      Ruleset="remote")
---
# rsyslog DaemonSet
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  name: rsyslog
  labels:
    component: rsyslog
    version: v1
spec:
  template:
    metadata:
      labels:
        component: rsyslog
        version: v1
  spec:
    hostNetwork: true
    containers:
      - name: rsyslog
        image: example/rsyslog
        imagePullPolicy: IfNotPresent
        volumeMounts:
          - name: rsyslog-config-volume
            mountPath: /etc/rsyslog.d
          - name: log-volume
            mountPath: /var/log
          - name: container-volume
            mountPath: /var/lib/docker/containers
    volumes:
      - name: rsyslog-config-volume
        configMap:
          name: rsyslog-config
      - name: log-volume
        hostPath:

```

```

    path: /var/log
  - name: container-volume
    hostPath:
      path: /var/lib/docker/containers

```

2 DaemonSet を作成します。

```
kubectl apply -f <daemonset yaml file>
```

例：

次のステップ

セキュリティの考慮事項

NCP を展開するときに、Kubernetes と NSX-T の両方の環境を保護する必要があります。

指定したノードでのみ実行するように NCP を制限する

NCP は、NSX-T 管理プレーンにアクセスできるため、指定されたインフラストラクチャ ノードでのみ実行するように制限する必要があります。これらのノードは、適切なラベルを付けて区別することができます。このラベルの nodeSelector を NCP ReplicationController の仕様に適用する必要があります。

```

nodeSelector:
  nsx-infra: True

```

アフィニティなどの他の方法でノードにポッドを割り当てることもできます。詳細については、<https://kubernetes.io/docs/concepts/configuration/assign-pod-node> を参照してください。

Docker Engine を最新の状態にする

Docker は、セキュリティ更新を定期的にリリースしています。これらの更新を自動的に適用する必要があります。

信頼できないコンテナの NET_ADMIN と NET_RAW 機能を許可しない

NET_ADMIN と NET_RAW の Linux 機能は、ポッド ネットワークに侵入した攻撃者によって悪用される可能性があります。信頼されていないコンテナでは、これらの 2 つの機能を無効にする必要があります。デフォルトでは、権限のないコンテナに NET_ADMIN 機能は付与されません。ポッドの仕様でこの機能が明示的に有効になったり、コンテナに特権モードが設定されないように注意してください。信頼されていないコンテナの場合、コンテナの仕様で SecurityContext 設定の削除機能リストに NET_RAW を指定し、NET_RAW を無効にします。次に例を示します。

```

securityContext:
  capabilities:
    drop:
      - NET_RAW
      - ...

```

ロールベースのアクセス制御

Kubernetes は、ロールベースのアクセス制御 (RBAC) API を使用して認証を処理します。管理者はポリシーを動的に設定することができます。詳細については、<https://kubernetes.io/docs/admin/authorization/rbac> を参照してください。

通常、特権アクセスとロールが付与されているのはクラスタ管理者だけです。ユーザー アカウントとサービス アカウントの場合には、アクセス権を付与するときに、最小限の権限を付与する必要があります。

推奨のガイドラインは次のとおりです。

- Kubernetes API トークンを必要とするポッドにのみトークンへのアクセスを許可します。
- NCP ConfigMap と NSX API クライアント証明書の TLS シークレットへのアクセスを NCP ポッドに限定します。
- このようなアクセスを必要としないポッドから Kubernetes ネットワーク API へのアクセスをブロックします。
- Kubernetes RBAC ポリシーを追加し、Kubernetes API にアクセスできるポッドを指定します。

NCP ポッドの推奨 RBAC ポリシー

ServiceAccount で NCP ポッドを作成し、このアカウントに最小の権限セットを付与します。また、他のポッドまたは ReplicationControllers には、NCP ReplicationController および NSX ノード エージェントのボリュームとしてマウントされる ConfigMap と TLS シークレットへのアクセスを許可しません。

次の例は、NCP にロールとロールのバインドを指定する方法を示しています。

```
Cluster wide role to read, watch and get resources
----
kind: ClusterRole
# Set the apiVersion to v1 if running with OpenShift
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: ncp-cluster-role
rules:
- apiGroups:
  - ""
  - extensions
  resources:
  - deployments
  - endpoints
  - pods
  - namespaces
  - networkpolicies
  - nodes
  - replicationcontrollers
  - services
  verbs:
  - get
  - watch
  - list
----
Cluster wide role to read, watch, get and modify ingresses
----
```

```

kind: ClusterRole
# Set the apiVersion to v1 if running with OpenShift
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: ncp-ingress-role
rules:
  - apiGroups:
    - ""
    - extensions
    resources:
    - ingresses
    verbs:
    - get
    - watch
    - list
    - update
    - patch
  - apiGroups:
    - extensions
    resources:
    - ingresses/status
    verbs:
    - replace
    - update
    - patch
----
Bind roles to ServiceAccount belonging to NCP
----
# Set the apiVersion to v1 if running with OpenShift
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: ncp-cluster-role-binding
roleRef:
  # Comment out the apiGroup if running with OpenShift
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: ncp-cluster-role
subjects:
  - kind: ServiceAccount
    name: ncp-svc-account
    namespace: ncp-deployed-ns-name
----
----
# Set the apiVersion to v1 if running with OpenShift
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: ncp-ingress-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: ncp-ingress-role
subjects:
  - kind: ServiceAccount

```

```
name: ncp-svc-account  
namespace: ncp-deployed-ns-name
```

注： NSX-T クライアント証明書に Kubernetes API を使用した作成された TLS シークレットとプライベート キーのペアには、Kubernetes API サーバにアクセスできる任意のポッドからアクセスできます。ポッドがサービス アカウントなしで作成された場合、同じ名前空間のデフォルトのサービス アカウントが自動的に割り当てられます。これにより、Kubernetes API にアクセスするトークンが自動的にマウントされます。これらのトークンへのアクセスは、トークンを必要とするポッドにのみ許可する必要があります。

NSX-T Container Plug-in の管理

4

NSX-T Container Plug-in は、NSX Manager グラフィカル ユーザー インターフェイスまたはコマンドライン インターフェイス (CLI) から管理できます。

この章には、次のトピックが含まれています。

- NSX Manager グラフィカル ユーザー インターフェイスでの IP ブロックの管理
- NSX Manager グラフィカル ユーザー インターフェイスでの IP ブロックのサブネットの管理
- CIF 接続論理ポート
- CLI コマンド

NSX Manager グラフィカル ユーザー インターフェイスでの IP ブロックの管理

NSX Manager グラフィカル ユーザー インターフェイスで、IP ブロックのタグを追加、削除、編集、表示、管理できます。詳細を確認することもできます。

手順

- 1 ブラウザから NSX Manager (<https://<nsx-manager-IP-address-or-domain-name>>) にログインします。
- 2 [DDI] を選択します。
既存の IP アドレスのブロックの一覧が表示されます。
- 3 次のいずれかのアクションを実行します。

オプション	アクション
IP ブロックを追加する	[追加] をクリックします。
1 つ以上の IP ブロックを削除する	1 つ以上の IP ブロックを選択し、[削除] をクリックします。
IP ブロックを編集する	IP ブロックを選択して [編集] をクリックします。
IP ブロックの詳細を表示する	IP ブロック名をクリックします。全般的な情報を表示するには、[概要] タブをクリックします。この IP ブロックのサブネットを表示するには、[サブネット] タブをクリックします。
IP ブロックのタグを管理します。	IP ブロックを選択し、[アクション] - [タグの管理] の順にクリックします。

サブネットが割り当てられた IP ブロックは削除できません。

NSX Manager グラフィカル ユーザー インターフェイスでの IP ブロックのサブネットの管理

NSX Manager グラフィカル ユーザー インターフェイスから IP ブロックにサブネットを追加したり、削除することができます。

手順

- 1 ブラウザから NSX Manager (<https://<nsx-manager-IP-address-or-domain-name>>) にログインします。
- 2 [DDI] を選択します。
既存の IP アドレスのブロックの一覧が表示されます。
- 3 IP ブロックの名前をクリックします。
- 4 [サブネット] タブをクリックします。
- 5 次のいずれかのアクションを実行します。

オプション	アクション
IP ブロック サブネットを追加する	[追加] をクリックします。
1 つ以上の IP ブロック サブネットを削除します。	1 つ以上のサブネットを選択して、[削除] をクリックします。

CIF 接続論理ポート

CIF (コンテナ インターフェイス) は、スイッチ上の論理ポートに接続されているコンテナのネットワーク インターフェイスです。これらのポートは CIF 接続論理ポートといいます。

NSX Manager グラフィカル ユーザー インターフェイスから CIF 接続論理ポートを管理できます。

CIF 接続論理ポートの管理

[スイッチング] - [ポート] の順に移動して、CIF 接続論理ポートを含むすべての論理ポートを表示します。CIF 接続論理ポートの接続リンクをクリックして、接続情報を表示します。論理ポートのリンクをクリックすると、ウィンドウ ペインが開き、[概要]、[監視]、[管理]、[関連] の 4 つのタブが表示されます。[関連] - [論理ポート] の順にクリックすると、アップリンク スイッチに関連する論理ポートが表示されます。スイッチ ポートの詳細については、『NSX-T 管理ガイド』を参照してください。

ネットワーク監視ツール

次のツールは、CIF 接続論理ポートをサポートします。これらのツールの詳細については、『NSX-T 管理ガイド』を参照してください。

- トレースフロー

- ポート接続
- IPFIX
- コンテナに接続している論理スイッチ ポートの GRE カプセル化を使用して、リモート ポート ミラーリングを実行できます。詳細については、『NSX-T 管理ガイド』の「ポート ミラーリング スイッチング プロファイルについて」を参照してください。ただし、CIF から VIF ポートへのポート ミラーリングはサポートされません。

今回のリリースでは、分散ネットワーク暗号化はサポートされていません。

CLI コマンド

CLI コマンドを実行するには、NSX-T Container Plug-in コンテナにログインしてターミナルを開き、`nsxcli` コマンドを実行します。

ノードで次のコマンドを実行しても、CLI プロンプトを表示できます。

```
kubectrl exec -it <pod name> nsxcli
```

表 4-1. NCP コンテナの CLI コマンド

タイプ	コマンド
ステータス	get ncp-nsx status
ステータス	get ncp-k8s-api-server status
ステータス	get ncp-watcher <watcher-name>
ステータス	get ncp-watchers
キャッシュ	get project-cache <project-name>
キャッシュ	get project-caches
キャッシュ	get namespace-cache <namespace-name>
キャッシュ	get namespace-caches
キャッシュ	get pod-cache <pod-name>
キャッシュ	get pod-caches
キャッシュ	get ingress-caches
サポート	get support-bundle file <filename>
サポート	get ncp-log file <filename>
サポート	get node-agent-log file <filename>
サポート	get node-agent-log file <filename> <node-name>

表 4-2. NSX ノード エージェント コンテナの CLI コマンド

タイプ	コマンド
ステータス	get node-agent-hyperbus status
キャッシュ	get app-cache <app-name>
キャッシュ	get app-caches

表 4-3. NSX Kube プロキシ コンテナの CLI コマンド

タイプ	コマンド
ステータス	get ncp-k8s-api-server status
ステータス	get kube-proxy-watcher <watcher-name>
ステータス	get kube-proxy-watchers
ステータス	dump ovs-flows

NCP コンテナのステータス コマンド

- NCP と NSX Manager の間の接続ステータスを表示します。

```
get ncp-nsx status
```

例 :

```
kubenode> get ncp-nsx status
NSX Manager status: Healthy
```

- NCP と Kubernetes API サーバ間の接続ステータスを表示します。

```
get ncp-k8s-api-server status
```

例 :

```
kubenode> get ncp-k8s-api-server status
Kubernetes ApiServer status: Healthy
```

- 入力側、名前空間、ポッド、サービスの監視ステータスを表示します。

```
get ncp-watcher <watcher-name>
get ncp-watchers
```

例 1 :

```
kubenode> get ncp-watcher pod
Average event processing time: 1174 msec (in past 3600-sec window)
Current watcher started time: Mar 02 2017 10:47:35 PST
Number of events processed: 1 (in past 3600-sec window)
Total events processed by current watcher: 1
Total events processed since watcher thread created: 1
Total watcher recycle count: 0
Watcher thread created time: Mar 02 2017 10:47:35 PST
Watcher thread status: Up
```

例 2 :

```
kubenode> get ncp-watchers
pod:
Average event processing time: 1145 msec (in past 3600-sec window)
Current watcher started time: Mar 02 2017 10:51:37 PST
```

```

Number of events processed: 1 (in past 3600-sec window)
Total events processed by current watcher: 1
Total events processed since watcher thread created: 1
Total watcher recycle count: 0
Watcher thread created time: Mar 02 2017 10:51:37 PST
Watcher thread status: Up

namespace:
Average event processing time: 68 msec (in past 3600-sec window)
Current watcher started time: Mar 02 2017 10:51:37 PST
Number of events processed: 2 (in past 3600-sec window)
Total events processed by current watcher: 2
Total events processed since watcher thread created: 2
Total watcher recycle count: 0
Watcher thread created time: Mar 02 2017 10:51:37 PST
Watcher thread status: Up

ingress:
Average event processing time: 0 msec (in past 3600-sec window)
Current watcher started time: Mar 02 2017 10:51:37 PST
Number of events processed: 0 (in past 3600-sec window)
Total events processed by current watcher: 0
Total events processed since watcher thread created: 0
Total watcher recycle count: 0
Watcher thread created time: Mar 02 2017 10:51:37 PST
Watcher thread status: Up

service:
Average event processing time: 3 msec (in past 3600-sec window)
Current watcher started time: Mar 02 2017 10:51:37 PST
Number of events processed: 1 (in past 3600-sec window)
Total events processed by current watcher: 1
Total events processed since watcher thread created: 1
Total watcher recycle count: 0
Watcher thread created time: Mar 02 2017 10:51:37 PST
Watcher thread status: Up

```

NCP コンテナのキャッシュ コマンド

- プロジェクトまたは名前空間の内部キャッシュを取得します

```

get project-cache <project-name>
get project-caches
get namespace-cache <namespace-name>
get namespace-caches

```

例 1 :

```

kubenode> get project-cache default
isolation:
  is_isolated: False
logical-router: 8acc9cd-9883-45f6-81b3-0d1fb2583180
logical-switch:

```

```
id: 9d7da647-27b6-47cf-9cdb-6e4f4d5a356d
ip_pool_id: 519ff57f-061f-4009-8d92-3e6526e7c17e
subnet: 10.0.0.0/24
subnet_id: f75fd64c-c7b0-4b42-9681-fc656ae5e435
```

例 2 :

```
kubenode> get project-caches
default:
  isolation:
    is_isolated: False
  logical-router: 8accc9cd-9883-45f6-81b3-0d1fb2583180
  logical-switch:
    id: 9d7da647-27b6-47cf-9cdb-6e4f4d5a356d
    ip_pool_id: 519ff57f-061f-4009-8d92-3e6526e7c17e
    subnet: 10.0.0.0/24
    subnet_id: f75fd64c-c7b0-4b42-9681-fc656ae5e435

kube-system:
  Isolation:
    is_isolated: False
  logical-router: 5032b299-acad-448e-a521-19d272a08c46
  logical-switch:
    id: 85233651-602d-445d-ab10-1c84096cc22a
    ip_pool_id: ab1c5b09-7004-4206-ac56-85d9d94bffa2
    subnet: 10.0.1.0/24
    subnet_id: 73e450af-b4b8-4a61-a6e3-c7ddd15ce751
```

■ ポッドの内部キャッシュを取得します

```
get pod-cache <pod-name>
get pod-caches
```

例 1 :

```
kubenode> get pod-cache nsx.default.nginx-rc-uq2lv
cif_id: 2af9f734-37b1-4072-ba88-abbf935bf3d4
gateway_ip: 10.0.0.1
ingress_controller: False
ip: 10.0.0.2/24
labels:
  app: nginx
mac: 02:50:56:00:08:00
port_id: d52c833a-f531-4bdf-bfa2-e8a084a8d41b
vlan: 1
```

例 2 :

```
kubenode> get pod-caches
nsx.default.nginx-rc-uq2lv:
  cif_id: 2af9f734-37b1-4072-ba88-abbf935bf3d4
  gateway_ip: 10.0.0.1
  ingress_controller: False
  ip: 10.0.0.2/24
```

```
labels:
  app: nginx
  mac: 02:50:56:00:08:00
  port_id: d52c833a-f531-4bdf-bfa2-e8a084a8d41b
  vlan: 1
```

- 入力側の内部キャッシュを取得します。

```
get ingress caches
```

例：

```
kubenode> get ingress-caches
nsx.default.nginx-ingress-rc-host-ed3og: 10.192.162.201
```

NCP コンテナのサポート コマンド

- ファイルストアに NCP サポート バンドルを保存します。

サポート バンドルは、ポッド内のすべてのコンテナのログ ファイルから構成され、**tier:nsx-networking** というラベルが付いています。バンドル ファイルは tgz 形式で、CLI のデフォルトのファイルストア ディレクトリ `/var/vmware/nsx/file-store` に保存されます。CLI `file-store` コマンドを使用すると、バンドル ファイルをリモート サイトにコピーできます。

```
get support-bundle file <filename>
```

例：

```
kubenode>get support-bundle file foo
Bundle file foo created in tgz format
kubenode>copy file foo url scp://nicira@10.0.0.1:/tmp
```

- NCP ログをファイルストアに保存します。

ログ ファイルは tgz 形式で、CLI のデフォルトのファイルストア ディレクトリ `/var/vmware/nsx/file-store` に保存されます。CLI `file-store` コマンドを使用すると、バンドル ファイルをリモート サイトにコピーできます。

```
get ncp-log file <filename>
```

例：

```
kubenode>get ncp-log file foo
Log file foo created in tgz format
```

- ノード エージェント ログをファイルストアに保存します。

1 台のノードまたはすべてのノードのノード エージェント ログを保存します。ログは tgz 形式で、CLI のデフォルトのファイルストア ディレクトリ `/var/vmware/nsx/file-store` に保存されます。CLI `file-store` コマンドを使用すると、バンドル ファイルをリモート サイトにコピーできます。

```
get node-agent-log file <filename>
get node-agent-log file <filename> <node-name>
```

例 :

```
kubenode>get node-agent-log file foo
Log file foo created in tgz format
```

NSX ノード エージェント コンテナのステータス コマンド

- このノードのノード エージェントと HyperBus 間の接続ステータスを表示します。

```
get node-agent-hyperbus status
```

例 :

```
kubenode> get node-agent-hyperbus status
HyperBus status: Healthy
```

NSX ノード エージェント コンテナのキャッシュ コマンド

- アプリケーションの内部キャッシュを取得します。特定のアプリケーションまたはすべてのアプリケーションのキャッシュを取得できます。

```
get app-cache <app-name>
get app-caches
```

例 1 :

```
kubenode> get app-cache cif104
ip: 192.168.0.14/32
mac: 50:01:01:01:01:14
gateway_ip: 169.254.1.254/16
vlan_id: 104
```

例 2 :

```
kubenode> get app-caches
cif104:
ip: 192.168.0.14/32
mac: 50:01:01:01:01:14
gateway_ip: 169.254.1.254/16
vlan_id: 104
```

NSX Kube-Proxy コンテナのステータス コマンド

- Kube プロキシと Kubernetes API サーバ間の接続ステータスを表示します。

```
get ncp-k8s-api-server status
```

例 :

```
kubenode> get kube-proxy-k8s-api-server status
Kubernetes ApiServer status: Healthy
```

- Kube プロキシの監視ステータスを表示します。

```
get kube-proxy-watcher <watcher-name>
get kube-proxy-watchers
```

例 1 :

```
kubenode> get kube-proxy-watcher endpoint
Average event processing time: 15 msec (in past 3600-sec window)
Current watcher started time: May 01 2017 15:06:24 PDT
Number of events processed: 90 (in past 3600-sec window)
Total events processed by current watcher: 90
Total events processed since watcher thread created: 90
Total watcher recycle count: 0
Watcher thread created time: May 01 2017 15:06:24 PDT
Watcher thread status: Up
```

例 2 :

```
kubenode> get kube-proxy-watchers
endpoint:
Average event processing time: 15 msec (in past 3600-sec window)
Current watcher started time: May 01 2017 15:06:24 PDT
Number of events processed: 90 (in past 3600-sec window)
Total events processed by current watcher: 90
Total events processed since watcher thread created: 90
Total watcher recycle count: 0
Watcher thread created time: May 01 2017 15:06:24 PDT
Watcher thread status: Up

service:
Average event processing time: 8 msec (in past 3600-sec window)
Current watcher started time: May 01 2017 15:06:24 PDT
Number of events processed: 2 (in past 3600-sec window)
Total events processed by current watcher: 2
Total events processed since watcher thread created: 2
Total watcher recycle count: 0
Watcher thread created time: May 01 2017 15:06:24 PDT
Watcher thread status: Up
```

■ ノードの OVS フローのダンプ

```
dump ovs-flows
```

例 :

```
kubenode> dump ovs-flows
NXST_FLOW reply (xid=0x4):
  cookie=0x0, duration=8.876s, table=0, n_packets=0, n_bytes=0, idle_age=8, priority=100,ip
actions=ct(table=1)
  cookie=0x0, duration=8.898s, table=0, n_packets=0, n_bytes=0, idle_age=8, priority=0
actions=NORMAL
  cookie=0x0, duration=8.759s, table=1, n_packets=0, n_bytes=0, idle_age=8,
priority=100,tcp,nw_dst=10.96.0.1,tp_dst=443 actions=mod_tp_dst:443
  cookie=0x0, duration=8.719s, table=1, n_packets=0, n_bytes=0, idle_age=8,
priority=100,ip,nw_dst=10.96.0.10 actions=drop
  cookie=0x0, duration=8.819s, table=1, n_packets=0, n_bytes=0, idle_age=8,
priority=90,ip,in_port=1 actions=ct(table=2,nat)
  cookie=0x0, duration=8.799s, table=1, n_packets=0, n_bytes=0, idle_age=8, priority=80,ip
actions=NORMAL
  cookie=0x0, duration=8.856s, table=2, n_packets=0, n_bytes=0, idle_age=8, actions=NORMAL
```