

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

変更日 : 2018 年 7 月 18 日

VMware NSX-T 2.2.1

VMware NSX-T Data Center 2.2



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](mailto:docfeedback@vmware.com) までお送りください。

VMware, Inc.  
3401 Hillview Ave.  
Palo Alto, CA 94304  
[www.vmware.com](http://www.vmware.com)

ヴァイエムウェア株式会社  
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](http://www.vmware.com/jp)

Copyright © 2017, 2018 VMware, Inc. All rights reserved. [著作権および商標情報](#)。

# 目次

## NSX-T Container Plug-in for Kubernetes and Cloud Foundry : インストールおよび管理ガイド 5

### 1 NSX-T Container Plug-in の概要 6

- 互換性の要件 7
- インストールの概要 8
- NCP のアップデート 8

### 2 NSX-T リソースのセットアップ 9

- NSX-T リソースの設定 9
- Tier-0 論理ルーターの作成と設定 12

### 3 Kubernetes 環境への NCP のインストール 14

- NSX-TCNI プラグインのインストール 14
- OVS のインストールと構成 15
- Kubernetes ノードの NSX-T ネットワークの設定 17
- NSX Node Agent のインストール 17
- nsx-node-agent-ds.yml の ncp.ini の ConfigMap 18
- NSX-T Container Plug-in のインストール 21
- ncp-rc.yml の ncp.ini の ConfigMap 23
- 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 26
- NCP ポッドでの証明書ファイルのマウント 27
- Syslog の設定 28
  - Syslog のサイドカー コンテナの作成 28
  - Syslog の DaemonSet レプリカの作成 30
  - 例：サイドカー コンテナで実行中のログ ローテーションおよび Syslog の設定 31
- セキュリティの考慮事項 38
- ネットワーク リソースの設定のヒント 41

### 4 Pivotal Cloud Foundry 環境への NCP のインストール 43

- Pivotal Cloud Foundry 環境への NCP のインストール 43

### 5 ロード バランシング 46

- ロード バランシングの設定 46

### 6 NSX-T Container Plug-in の管理 50

- 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 50
- 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 51

CIF 接続論理ポート 51

CLI コマンド 52

エラー コード 70

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

このガイドでは、NSX-T および Kubernetes 間、また、NSX-T および Pivotal Cloud Foundry (PCF) 間の連携を実現するために、NSX-T Container Plug-in (NCP) をインストールおよび管理する方法について説明します。

## 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。NSX-T、Kubernetes、および Pivotal Cloud Foundry のインストールと管理について理解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。VMware 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 や Pivotal Cloud Foundry などのコンテナベースの Platform as a Service (PaaS) 製品との連携を可能にします。このガイドでは、Kubernetes と Pivotal Cloud Foundry を使用した NCP の設定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 制御プレーンとの通信を行います。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。

NCP では、次の機能が提供されます。

- Kubernetes クラスタに NSX-T 論理トポロジを自動的に作成し、Kubernetes の個々の名前空間に論理ネットワークを作成します。
- Kubernetes のポッドを論理ネットワークに接続し、IP アドレスと MAC アドレスを割り当てます。
- ネットワーク アドレス変換 (NAT) がサポートされているため、Kubernetes の各名前空間に個別の SNAT IP を割り当てることができます。

---

**注：** NAT を設定する場合、変換される IP アドレスの合計数は 1,000 を超えることはできません。

---

- NSX-T 分散ファイアウォールを使用した Kubernetes ネットワーク ポリシーを実装。
  - 入力方向および出力方向でネットワーク ポリシーをサポート。
  - ネットワーク ポリシーで IPBlock セレクタをサポート。
  - ネットワーク ポリシーでラベル セレクタを指定する際に `matchLabels` と `matchExpression` をサポート。
- Kubernetes のタイプ `ClusterIP` のサービスおよびタイプ `LoadBalancer` のサービスを実装。
- NSX-T レイヤー 7 ロード バランサを使用する Kubernetes Ingress を実装。
  - HTTP Ingress と、TLS Edge ターミネーション使用する HTTPS Ingress をサポート。
  - Ingress のデフォルトのバックエンド構成をサポート。

- Ingress の URI の書き換えをサポート。
- 名前空間、ポッド名、およびポッドのラベル用のタグを NSX-T 論理スイッチ ポート上に作成し、管理者がタグベースで NSX-T のセキュリティ グループとポリシーを定義できるようにします。

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。同じ NSX-T 環境を使用して、複数の Kubernetes クラスタに、異なる NCP インスタンスを配置することができます。

この章には、次のトピックが含まれています。

- [互換性の要件](#)
- [インストールの概要](#)
- [NCP のアップデート](#)

## 互換性の要件

NSX-T Container Plug-in(NCP) で Kubernetes 環境および Pivotal Cloud Foundry (PCF) 環境との互換性を実現するには、次の要件を満たす必要があります。

表 1-1. Kubernetes 環境の互換性要件

ソフトウェア製品	バージョン
NSX-T	2.1、2.2
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"> <li>■ <a href="#">サポート対象の vSphere バージョン</a></li> <li>■ RHEL KVM 7.4、7.5</li> <li>■ Ubuntu KVM 16.04</li> </ul>
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"> <li>■ RHEL 7.4、7.5</li> <li>■ Ubuntu 16.04</li> </ul>
コンテナ ランタイム	Docker 1.13
Container Orchestrator	<ul style="list-style-type: none"> <li>■ Kubernetes 1.9、1.10</li> <li>■ Kubernetes 1.11 (NCP 2.2.1 とともに使用)</li> </ul>
コンテナ ホストの vSwitch	OVS 2.8.1 (NSX-T 2.1 に含まれています)、2.9.1 (NSX-T 2.2 に含まれています)

表 1-2. Cloud Foundry 環境の互換性要件

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"> <li>■ <a href="#">サポート対象の vSphere バージョン</a></li> <li>■ RHEL KVM 7.4、7.5</li> <li>■ Ubuntu KVM 16.04</li> </ul>
Container Orchestrator	<ul style="list-style-type: none"> <li>■ Pivotal Application Service 2.1.x (2.1.0 を除く) と Pivotal Operations Manager 2.1.x</li> <li>■ Pivotal Application Service 2.2.0 および Pivotal Operations Manager 2.2.0</li> </ul>

## 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 のインストールと設定を行います。手順を正常に実行するには、NSX-T および Kubernetes のインストールと管理について理解している必要があります。

- 1 NSX-T をインストールします。
- 2 オーバーレイ トランスポート 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 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 ネットワークを設定します。
- 10 DaemonSet として NSX Node Agent をインストールします。
- 11 Replication Controller 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。

## NCP のアップデート

NCP を 2.2.0 にアップデートするには、次の手順を実行してください。

- 1 Pivotal Application Service (PAS) 2.0.x を実行している場合、最初に PAS を 2.1.x にアップデートします。
- 2 NCP または NSX-TTile をバージョン 2.2.0 にアップデートします。
- 3 NSX-T を 2.2 にアップデートします。

# 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 プールなどがあります。

設定ファイル `ncp.ini` で、UUID または名前を使用して NSX-T リソースを設定します。

### オーバーレイ トランスポート ゾーン

NSX Manager にログインし、[ファブリック] - [トランスポート ゾーン] の順に移動します。コンテナ ネットワーク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。

`ncp.ini` の `[nsx_v3]` セクションで `overlay_tz` オプションを設定して、クラスタのオーバーレイ トランスポート ゾーンを指定します。

### Tier-0 論理ルーティング

NSX Manager にログインし、[ルーティング] - [ルーター] の順に移動します。コンテナ ネットワークに使用されているルーターを検索するか、新規に作成します。

`ncp.ini` の `[nsx_v3]` セクションで `tier0_router` オプションを設定して、クラスタの Tier-0 論理ルーターを指定します。

---

**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

ノードがデータトラフィック用に使用する vNIC 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、接続するとセットアップが容易になります。NSX Manager にログインし、[スイッチング] - [スイッチ] の順に移動して、論理スイッチを作成できます。スイッチ上で論理ポートを作成し、ノードの vNIC を接続します。論理ポートには次のタグが必要です。

- スコープ : ncp/cluster、タグ : <cluster\_name>
- スコープ : ncp/node\_name、タグ : <node\_name>

<cluster\_name> 値は、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。

## Kubernetes ポッドの IP アドレス ブロック

NSX Manager にログインして、[DDI] - [IP アドレス管理] の順に移動し、1 つ以上の 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。

ncp.ini の [nsx\_v3] セクションで container\_ip\_blocks オプションを設定して、Kubernetes ポッドの IP アドレス ブロックを指定します。

また、SNAT 以外の名前空間（Kubernetes 用）またはクラスタ（PCF 用）専用の IP アドレス ブロックを作成することもできます。

ncp.ini の [nsx\_v3] セクションで no\_snat\_ip\_blocks オプションを設定して、SNAT 以外の IP アドレス ブロックを指定します。

NCP の実行中に SNAT 以外の IP アドレス ブロックを作成した場合には、NCP を再起動する必要があります。再起動しない場合、IP アドレス ブロックが枯渇するまで、NCP は共有の IP アドレス ブロックを使用し続けます。

---

**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 プール

SNAT ルールを使用してポッドの IP アドレスを変換する際、または SNAT/DNAT ルールを使用して Ingress Controller を公開する際に、IP アドレス プールは OpenStack のフローティング IP アドレスと同様に、IP アドレスの割り当てに使用されます。これらの IP アドレスは、外部 IP アドレスとも呼ばれます。

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と同じサブネット 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。

NSX Manager にログインし、[インベントリ] - [グループ] - [IP アドレス プール] の順に移動して、プールを作成するか、既存のプールを検索します。

ncp.ini の [nsx\_v3] セクションで external\_ip\_pools オプションを設定して、SNAT の IP アドレス プールを指定します。

サービスにアノテーションを追加して、特定のサービスの SNAT を設定することもできます。次はその例です。

```
apiVersion: v1
kind: Service
metadata:
  name: svc-example
  annotations:
    ncp/snat_pool: <external IP pool ID or name>
  selector:
    app: example
...
```

NCP によって、このサービスの SNAT ルールが設定されます。このルールの送信元 IP アドレスは、一連のバックエンド ポッドです。宛先 IP アドレスは、指定した外部 IP アドレス プールから割り当てられた SNAT IP アドレスです。次の点に注意してください。

- `ncp/snat_pool` で指定されたプールは、サービスを設定する前に NSX-T に配置しておく必要があります。
- NSX-T では、サービスの SNAT ルールの優先順位はプロジェクトよりも高くなります。
- ポッドに複数の SNAT ルールが設定されている場合は、1 つのルールのみが機能します。

NCP はデフォルトで、PAS (Pivotal Application Service) 組織用に SNAT の IP アドレスを設定します。SNAT の IP アドレス プールの情報が含まれた `manifest.xml` を使用してアプリケーションを作成し、特定のアプリケーション用に SNAT IP アドレスを設定できます。次はその例です。

```
applications:
- name: frontend
  memory: 32M
  disk_quota: 32M
  buildpack: go_buildpack
  env:
    GOPACKAGENAME: example-apps/cats-and-dogs/frontend
    NCP_SNAT_POOL: <external IP pool ID or name>
...
```

NCP によって、このアプリケーションに SNAT ルールが設定されます。ルールの送信元 IP アドレスは一連のインスタンスの IP アドレスで、宛先 IP アドレスは、外部 IP アドレス プールから割り当てられた SNAT の IP アドレスです。次の点に注意してください。

- `NCP_SNAT_POOL` で指定されたプールは、アプリケーションのプッシュ前に NSX-T に配置しておく必要があります。
- アプリケーションの SNAT ルールの方が、組織の優先順位よりも高くなります。
- アプリケーションに設定できる SNAT の IP アドレスは、1 つのみです。

NCP (NSX Container Plug-in) 2.2.1 以降のリリースでは、次のタグを IP アドレス プールに追加することで、SNAT の IP アドレス プールから IP アドレスを割り当てる Kubernetes 名前空間または PCF 組織を指定できます。

- Kubernetes 名前空間の場合。スコープ : `ncp/owner`、タグ : `ns:<namespace_UUID>`
- PCF 組織の場合。スコープ : `ncp/owner`、タグ : `org:<org_UUID>`

名前空間または組織の UUID は、次のいずれかのコマンドで取得できます。

```
kubect1 get ns -o yaml
cf org <org_name> --guid
```

次の点に注意してください。

- 各タグには 1 つの UUID を指定する必要があります。同じプールに対して複数のタグを作成できます。
- 古いタグ ベースで一部の名前空間または組織に IP アドレスが割り当てられた後にタグを変更すると、Kubernetes サービスまたは PCF アプリケーションの SNAT 設定が変更されるか NCP が再起動されるまで、割り当てられた IP アドレスは再利用されません。
- IP アドレス プールに所有者タグがない場合は、すべての Kubernetes サービスまたは PCF アプリケーションにプールから IP アドレスを割り当てることができます。

## (オプション) (Kubernetes のみ) ファイアウォール マーカー セクション

管理者が作成したファイアウォール ルールが、ネットワーク ポリシーに基づいて NCP で作成したファイアウォール セクションに干渉されないようにするには、NSX Manager にログインし、[ファイアウォール] - [全般] の順に移動して、2 つのファイアウォール セクションを作成します。

ncp.ini の [nsx\_v3] セクションで bottom\_firewall\_section\_marker および top\_firewall\_section\_marker オプションを設定して、マーカー ファイアウォール セクションを指定します。

最下位のファイアウォール セクションは、最上位のファイアウォール セクションの下に配置される必要があります。これらのファイアウォール セクションを作成することで、NCP が分離用に作成するすべてのファイアウォール セクションは、リストの最下位のファイアウォール セクションの上に作成され、NCP がポリシー用に作成するすべてのファイアウォール セクションは、最上位のファイアウォール セクションの下に作成されるようになります。マーカー セクションを作成しない場合、すべての分離ルールはリストの最下位に作成され、すべてのポリシー セクションは最上位に作成されます。同じ値を持つ複数のマーカー ファイアウォール セクションを 1 つのクラスタに設定することはできないため、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] - [ルーター] の順に移動して、[追加] - [Tier-0 ルーター] 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] 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] - [ルート再配分] 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] をクリックします。

送信元の場合には、ルーティング (NAT ではない) トポロジで [NSX スタティック] 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] 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] - [ルーター ポート] の順にクリックします。

- 13 [追加] 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] 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。

# Kubernetes 環境への NCP 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-TCNI 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 Node Agent 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
- ネットワーク リソースの設定のヒント

## NSX-TCNI 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する必要があります。

Ubuntu の場合、NSX-T CNI プラグインをインストールすると、AppArmor プロファイル ファイル `ncp-apparmor` が `/etc/apparmor.d` にコピーされ、ロードされます。インストールの前に、AppArmor サービスが実行されており、`/etc/apparmor.d` ディレクトリを作成しておく必要があります。これを行わないと、インストールは失敗します。次のコマンドで、AppArmor モジュールが有効かどうかを確認できます。

```
sudo cat /sys/module/apparmor/parameters/enabled
```

次のコマンドで、AppArmor サービスが開始されているかどうかを確認できます。

```
sudo /etc/init.d/apparmor status
```

NSX-T CNI プラグインのインストール時に AppArmor サービスを実行していない場合、インストールが終了するとメッセージが表示されます。

```
subprocess installed post-installation script returned error exit status 1
```

このメッセージには、AppArmor プロファイルのロードを除いて、すべてのインストール手順が完了したことを示されます。

ncp-apparmor プロファイル ファイルは、node-agent-apparmor と呼ばれる NSX Node Agent 用の AppArmor プロファイルを提供します。これは、docker-default プロファイルとは次の点で異なります。

- deny mount ルールが削除されている。
- mount ルールが追加されている。
- いくつかの network、capability、file、および umount オプションが追加されている。

node-agent-apparmor プロファイルは、別のプロファイルに置き換えることができます。ただし、プロファイル名「node-agent-apparmor」は、NSX Node Agent のインストールに使用する nsx-node-agent-ds.yml ファイルで参照されます。別のプロファイルを使用する場合は、nsx-node-agent-ds.yml のセクション spec:template:metadata:annotations にある次のエントリでプロファイル名を指定する必要があります。

```
container.apparmor.security.beta.kubernetes.io/<container-name>: localhost/<profile-name>
```

#### 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。

- 2 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

- 1 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。

- 2 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。
- 3 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
```

- 4 OVS が実行されていることを確認します。

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

- 5 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
```

- 6 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。

- 7 *br-int* と *node-if* link 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ともかまいません。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 Node Agent のインストール

NSX Node Agent 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 Node Agent 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 Node Agent 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.yml` 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.yml` 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。

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 次のコマンドを使用して、NSX Node Agent 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-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.
    # 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.yml` 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.yml` 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 クラスターと 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 サーバ証明書の確認に使用する 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-----
```

Ingress Controller を NAT モードで実行するように設定するには、`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用に 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 ファイルでは、これはすでに指定されています。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
```

**注：** NCP は、Kubernetes リソースのライフ サイクル イベントを監視するために、Kubernetes API サーバにパーシステント HTTP 接続を確立します。API サーバの障害またはネットワーク障害が原因で NCP の TCP 接続が無効になった場合は、NCP を再起動して、API サーバへの接続を再度確立する必要があります。接続できないと、NCP で新しいイベントが認識されません。

NCP ReplicationController のローリング アップデート中は、コンテナ ホストを再起動しないでください。何らかの理由でホストを再起動した場合、再起動後に 2 つの NCP ポッドが実行されることがあります。その場合は、次の操作を行います。

- いずれかの NCP ポッドを削除します。どちらを削除しても構いません。次はその例です。

```
kubectl delete pods <NCP pod name> -n nsx-system
```

- 名前空間 `nsx-system` を削除します。次はその例です。

```
kubectl delete -f ncp-rc.yml -n nsx-system
```

## 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-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.
    # 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>

# User name for the NSX manager (string value)
#nsx_api_user = <None>

# Password for the NSX manager (string value)
#nsx_api_password = <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

# Option to use native loadbalancer support.
#use_native_loadbalancer = False

# Option to set load balancing algorithm in load balancer pool object.
# Available choices are
# ROUND_ROBIN/LEAST_CONNECTION/IP_HASH
#pool_algorithm = 'ROUND_ROBIN'

# Option to set load balancer service size. Available choices are
# SMALL/MEDIUM/LARGE.
# MEDIUM Edge VM (4 vCPU, 8GB) only supports SMALL LB.
# LARGE Edge VM (8 vCPU, 16GB) only supports MEDIUM and SMALL LB.
# Bare Metal Edge (IvyBridge, 2 socket, 128GB) supports LARGE, MEDIUM and
# SMALL LB
#service_size = 'SMALL'

# Max number of virtual servers allowed on a single load balancer service.
# SMALL LB supports 10 virtual servers. MEDIUM LB supports 100 virtual servers.
# LARGE LB supports 1000 virtual servers'
# Defaults to 10.
#virtual_servers_per_lbs = 10

# Retrieve the node VIF via the display name of the VM, in deployments where
# the display name of the VM is same as the node name
#get_node_vif_by_vm_name = 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-ujo/nsx-cert
      readOnly: true
  volumes:
    ...
    - name: nsx-cert
      secret:
        secretName: SECRET_NAME

```

- 3 yml 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-ujo/nsx-cert/tls.crt
nsx_api_private_key_file = /etc/nsx-ujo/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-ujo/nsx_cert
          - name: nsx-priv-key
            # Mount path must match nsx_v3 option "nsx_api_private_key_file"
            mountPath: /etc/nsx-ujo/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 サイドカー 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、サイドカー 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 Node Agent を設定します。

NCP と NSX Node Agent 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。次はその例です。

```
[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: /var/log/nsx-ujo
```

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 ホストパスが配置され、ユーザー `nsx-ncp` によって書き込み可能であることを確認してください。

### a 次のコマンドを実行します。

```
mkdir -p <host-filesystem-log-dir-path>
chmod +w <host-filesystem-log-dir-path>
```

### b ユーザー `nsx-ncp` を追加するか、ホストパスのモードを `777` に変更します。

```
useradd -s /bin/bash nsx-ncp
chown nsx-ncp:nsx-ncp <host-filesystem-log-dir-path>
or
chmod 777 <host-filesystem-log-dir-path>
```

## 3 NCP ポッド仕様の yaml ファイルに、Syslog の ConfigMap を追加します。次はその例です。

```
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")
```

## 4 NCP ポッドの yaml ファイルに rsyslog コンテナに追加します。rsyslog が設定データを検索でき、他のコンテナからログを読み取り可能なボリュームをマウントします。次はその例です。

```
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。次はその例です。

```

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：サイドカー コンテナで実行中のログ ローテーションおよび Syslog の設定

次の手順では、サイドカー コンテナで実行中のログ ローテーションおよび Syslog を設定する方法を示します。

## ログ ディレクトリの作成とログ ローテーションの設定

- マスター ノードを含むすべてのノード上にログ ディレクトリを作成し、その所有者を、ID が 1000 のユーザーに変更します。

```
mkdir /var/log/nsx-ujo
chown localadmin:localadmin /var/log/nsx-ujo
```

- すべてのノードで、/var/log/nsx-ujo ディレクトリに対するログ ローテーションを設定します。

```
cat <<EOF > /etc/logrotate.d/nsx-ujo
/var/log/nsx-ujo/*.log {
    copytruncate
    daily
    size 100M
    rotate 4
    delaycompress
    compress
    notifempty
    missingok
}
EOF
```

## NCP レプリケーション コントローラの作成

- NCP 用に ncp.ini ファイルを作成します。

```
cat <<EOF > /tmp/ncp.ini
[DEFAULT]
log_dir = /var/log/nsx-ujo
[coe]
cluster = k8s-cl1
[k8s]
apiserver_host_ip = 10.114.209.77
apiserver_host_port = 6443
ca_file = /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
client_token_file = /var/run/secrets/kubernetes.io/serviceaccount/token
insecure = True
ingress_mode = nat
[nsx_v3]
nsx_api_user = admin
nsx_api_password = Password1!
nsx_api_managers = 10.114.209.68
insecure = True
subnet_prefix = 29
[nsx_node_agent]
[nsx_kube_proxy]
ovs_uplink_port = ens192
EOF
```

- ini ファイルから設定マップを作成します。

```
kubectl create configmap nsx-ncp-config-with-logging --from-file=/tmp/ncp.ini
```

- NCP rsyslog 構成ファイルを作成します。

```
cat <<EOF > /tmp/nsx-ncp-rsyslog.conf
# yaml template for NCP ReplicationController
# Correct kubernetes API and NSX API parameters, and NCP Docker image
# must be specified.
apiVersion: v1
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.licf.vmware.com"
        Port="514")

      stop
    }

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

- 上記の構成ファイルから、構成マップを作成します。

```
kubectl create -f /tmp/nsx-ncp-rsyslog.conf
```

- rsyslog サイドカー を使用して NCP レプリケーション コントローラを作成します。

```
cat <<EOF > /tmp/ncp-rc-with-logging.yml
# Replication Controller yaml for NCP
apiVersion: v1
kind: ReplicationController
metadata:
  # VMware NSX Container Plugin
  name: nsx-ncp
  labels:
    tier: nsx-networking
    component: nsx-ncp
    version: v1
spec:
  # Active-Active/Active-Standby is not supported in current release.
  # so replica *must be* 1.
  replicas: 1
  template:
    metadata:
```

```

labels:
  tier: nsx-networking
  component: nsx-ncp
  version: v1
spec:
  # NCP shares the host management network.
  hostNetwork: true
  nodeSelector:
    kubernetes.io/hostname: k8s-master
  tolerations:
    - key: "node-role.kubernetes.io/master"
      operator: "Exists"
      effect: "NoSchedule"
  containers:
    - name: nsx-ncp
      # Docker image for NCP
      image: nsx-ujo-docker-local.artifactory.eng.vmware.com/nsx-ncp:ob-6236425
      imagePullPolicy: IfNotPresent
      readinessProbe:
        exec:
          command:
            - cat
            - /tmp/ncp_ready
          initialDelaySeconds: 5
          periodSeconds: 5
          failureThreshold: 5
      securityContext:
        capabilities:
          add:
            - NET_ADMIN
            - SYS_ADMIN
            - SYS_PTRACE
            - DAC_READ_SEARCH
      volumeMounts:
        - name: config-volume
          # NCP expects ncp.ini is present in /etc/nsx-ujo
          mountPath: /etc/nsx-ujo
        - name: log-volume
          mountPath: /var/log/nsx-ujo
    - name: rsyslog
      image: jumanjiman/rsyslog
      imagePullPolicy: IfNotPresent
      volumeMounts:
        - name: rsyslog-config-volume
          mountPath: /etc/rsyslog.d
          readOnly: true
        - name: log-volume
          mountPath: /var/log/nsx-ujo
  volumes:
    - name: config-volume
      # ConfigMap nsx-ncp-config is expected to supply ncp.ini
      configMap:
        name: nsx-ncp-config-with-logging
    - name: rsyslog-config-volume
      configMap:

```

```

        name: rsyslog-config
      - name: log-volume
        hostPath:
          path: /var/log/nsx-ujo/
EOF

```

- 上記の仕様に基づいて NCP を作成します。

```
kubectl apply -f /tmp/ncp-rc-with-logging.yml
```

## NSX Node Agent DaemonSet の作成

- Node Agent の rsyslog 設定を作成します。

```

cat <<EOF > /tmp/nsx-node-agent-rsyslog.conf
# yaml template for NCP ReplicationController
# Correct kubernetes API and NSX API parameters, and NCP Docker image
# must be specified.
apiVersion: v1
kind: ConfigMap
metadata:
  name: rsyslog-config-node-agent
  labels:
    version: v1
data:
  ncp.conf: |
    module(load="imfile")

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

      stop
    }

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

    input(type="imfile"
      File="/var/log/nsx-ujo/nsx_node_agent.log"
      Tag="nsx_node_agent"
      Ruleset="remote")
EOF

```

- 上記から configmap を作成します。

```
kubectl create -f /tmp/nsx-node-agent-rsyslog.conf
```

- configmap サイドカー を使用して DaemonSet を作成します。

```
cat <<EOF > /tmp/nsx-node-agent-rsyslog.yml
# nsx-node-agent DaemonSet
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  name: nsx-node-agent
  labels:
    tier: nsx-networking
    component: nsx-node-agent
    version: v1
spec:
  template:
    metadata:
      annotations:
        container.apparmor.security.beta.kubernetes.io/nsx-node-agent: localhost/node-agent-
apparmor
    labels:
      tier: nsx-networking
      component: nsx-node-agent
      version: v1
  spec:
    hostNetwork: true
    tolerations:
      - key: "node-role.kubernetes.io/master"
        operator: "Exists"
        effect: "NoSchedule"
    containers:
      - name: nsx-node-agent
        # Docker image for NCP
        image: nsx-ujo-docker-local.artifactory.eng.vmware.com/nsx-ncp:ob-6236425
        imagePullPolicy: IfNotPresent
        # override NCP image entrypoint
        command: ["nsx_node_agent"]
        livenessProbe:
          exec:
            command:
              - /bin/sh
              - -c
              - ps aux | grep [n]sx_node_agent
            initialDelaySeconds: 5
            periodSeconds: 5
          securityContext:
            capabilities:
              add:
                - NET_ADMIN
                - SYS_ADMIN
                - SYS_PTRACE
                - DAC_READ_SEARCH
        volumeMounts:
          # ncp.ini
          - name: config-volume
            mountPath: /etc/nsx-ujo
          # mount openvswitch dir
```

```

- name: openvswitch
  mountPath: /var/run/openvswitch
# mount CNI socket path
- name: cni-sock
  mountPath: /var/run/nsx-ujo
# mount container namespace
- name: netns
  mountPath: /var/run/netns
# mount host proc
- name: proc
  mountPath: /host/proc
  readOnly: true
- name: log-volume
  mountPath: /var/log/nsx-ujo
- name: nsx-kube-proxy
# Docker image for NCP
image: nsx-ujo-docker-local.artifactory.eng.vmware.com/nsx-ncp:ob-6236425
imagePullPolicy: IfNotPresent
# override NCP image entrypoint
command: ["nsx_kube_proxy"]
livenessProbe:
  exec:
    command:
      - /bin/sh
      - -c
      - ps aux | grep [n]sx_kube_proxy
  initialDelaySeconds: 5
  periodSeconds: 5
securityContext:
  capabilities:
    add:
      - NET_ADMIN
      - SYS_ADMIN
      - SYS_PTRACE
      - DAC_READ_SEARCH
volumeMounts:
# ncp.ini
- name: config-volume
  mountPath: /etc/nsx-ujo
# mount openvswitch dir
- name: openvswitch
  mountPath: /var/run/openvswitch
- name: log-volume
  mountPath: /var/log/nsx-ujo
- name: rsyslog
  image: jumanjiman/rsyslog
  imagePullPolicy: IfNotPresent
  volumeMounts:
    - name: rsyslog-config-volume
      mountPath: /etc/rsyslog.d
      readOnly: true
    - name: log-volume
      mountPath: /var/log/nsx-ujo
volumes:
- name: config-volume

```

```

    configMap:
      name: nsx-ncp-config-with-logging
  - name: cni-sock
    hostPath:
      path: /var/run/nsx-ujo
  - name: netns
    hostPath:
      path: /var/run/netns
  - name: proc
    hostPath:
      path: /proc
  - name: openvswitch
    hostPath:
      path: /var/run/openvswitch
  - name: rsyslog-config-volume
    configMap:
      name: rsyslog-config-node-agent
  - name: log-volume
    hostPath:
      path: /var/log/nsx-ujo/
EOF

```

- DaemonSet を作成します。

```
kubectl apply -f /tmp/nsx-node-agent-rsyslog.yml
```

## 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。次はその例です。

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

## ロールベースのアクセス コントロール

Kubernetes は、ロールベースのアクセス コントロール (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 Secret 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 Node Agent のボリュームとしてマウントされる ConfigMap と TLS Secret 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。

```
# Create a ServiceAccount for NCP namespace
apiVersion: v1
kind: ServiceAccount
metadata:
  name: ncp-svc-account
  namespace: nsx-system

---

# Create ClusterRole for NCP
kind: ClusterRole
# Set the apiVersion to v1 while using OpenShift
```

```

apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: ncp-cluster-role
rules:
- apiGroups:
  - ""
  - extensions
  - networking.k8s.io
resources:
  - deployments
  - endpoints
  - pods
  - pods/log
  - namespaces
  - networkpolicies
  # Move 'nodes' to ncp-patch-role when hyperbus is disabled.
  - nodes
  - replicationcontrollers
  # Remove 'secrets' if not using Native Load Balancer.
  - secrets
verbs:
  - get
  - watch
  - list

```

```

---

# Create ClusterRole for NCP to edit resources
kind: ClusterRole
# Set the apiVersion to v1 while using OpenShift
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: ncp-patch-role
rules:
- apiGroups:
  - ""
  - extensions
resources:
  - ingresses
  - services
verbs:
  - get
  - watch
  - list
  - update
  - patch
- apiGroups:
  - ""
  - extensions
resources:
  - ingresses/status
  - services/status
verbs:
  - replace
  - update

```

```

- patch
---

# Bind ServiceAccount created for NCP to its ClusterRole
kind: ClusterRoleBinding
# Set the apiVersion to v1 while using OpenShift
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: ncp-cluster-role-binding
roleRef:
# Comment out the apiGroup while using OpenShift
apiGroup: rbac.authorization.k8s.io
kind: ClusterRole
name: ncp-cluster-role
subjects:
- kind: ServiceAccount
  name: ncp-svc-account
  namespace: nsx-system
---

# Bind ServiceAccount created for NCP to the patch ClusterRole
kind: ClusterRoleBinding
# Set the apiVersion to v1 while using OpenShift
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: ncp-patch-role-binding
roleRef:
# Comment out the apiGroup while using OpenShift
apiGroup: rbac.authorization.k8s.io
kind: ClusterRole
name: ncp-patch-role
subjects:
- kind: ServiceAccount
  name: ncp-svc-account
  namespace: nsx-system

```

**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 Secret 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。

## ネットワーク リソースの設定のヒント

ネットワーク リソースを設定する際は、特定の制限があることに注意してください。

### NSX-T のタグ付けの制限

NSX-T には、オブジェクトへのタグ付けに次の制限があります。

- スコープには、20 文字の制限があります。

- タグには、40 文字の制限があります。
- 各オブジェクトには、15 個までタグを付けることができます。

Kubernetes または OpenShift のアノテーションが NSX-T のスコープやタグにコピーされる際に字数の制限を超えると、問題が生じる場合があります。たとえば、スイッチ ポート用のタグがあり、これがファイアウォール ルールで使用されている場合、アノテーションのキーや値が字数制限を超えると、スコープまたはタグにすべてコピーされないため、ルールが想定したように適用されないことがあります。

## ネットワーク ポリシーの設定

ネットワーク ポリシーは、ラベル セレクタを使用してポッドまたは名前空間を選択します。

NCP がサポートする ネットワーク ポリシーは Kubernetes と同じであり、Kubernetes のバージョンによって異なります。

- Kubernetes 1.10 以前 : ネットワーク ポリシーのルール句に、`namespaceSelector`、`podSelector`、および `ipBlock` のセレクタが最大で 1 つ含まれている場合があります。
- Kubernetes 1.7 : 出力方向のルールおよび `ipBlock` CIDR セレクタはサポートされません。これらは、Kubernetes 1.8 以降でベータ版フィールドとして提供されます。

# Pivotal Cloud Foundry 環境への NCP のインストール

# 4

Pivotal Cloud Foundry (PCF) は、オープンソースの Platform as a Service (PaaS) プロバイダです。PCF 環境に NSX-T Container Plug-in (NCP) をインストールして、ネットワーク サービスを提供できます。

Pivotal Ops Manager を通じて作成された仮想マシンが NSX-T の機能を利用するには、コンテナ ネットワークへのレイヤー 3 接続が必要です。

高可用性 (HA) が自動的に有効になります。NCP をインストールする前に、次のタグを使用して、SpoofGuard のスニッチング プロファイルを作成する必要があります。

- スコープ: `ncp/cluster`、タグ: `<cluster_name>`
- スコープ: `ncp/ha`、タグ: `true`

**注:** セキュリティ グループに変更を加えたら、セキュリティ グループを適用しているすべてのアプリケーションを再度ステージングする必要があります。ステージングは、アプリケーションが実行されている場所にセキュリティ グループが適用されている場合、またはセキュリティ グループがグローバルに適用されている場合に行います。

この章には、次のトピックが含まれています。

- [Pivotal Cloud Foundry 環境への NCP のインストール](#)

## Pivotal Cloud Foundry 環境への NCP のインストール

NCP は、Pivotal Ops Manager のグラフィカル ユーザー インターフェイス (GUI) を介してインストールされます。

### 前提条件

Pivotal Ops Manager、NSX-T、および Pivotal Application Service (PAS) の新規インストール。必ず Pivotal Ops Manager を最初にインストールしてから、NSX-T、PAS の順にインストールしてください。詳細については、Pivotal Cloud Foundry のドキュメントを参照してください。

### 手順

- 1 Pivotal Cloud Foundry 用に NCP 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 `VMware-NSX-T.<version>.<build>.pivotal` です。
- 2 Pivotal Ops Manager に管理者としてログインします。

- 3 [製品のインポート] をクリックします。
- 4 ダウンロードしたファイルを選択します。
- 5 [Ops Manager Director for VMware vSphere] をクリックします。
- 6 [vCenter Server 設定] の [設定] タブで [NSX ネットワーク] を選択し、[NSX モード] で [NSX-T] を選択します。
- 7 [NSX CA 証明書] フィールドで、PEM 形式の証明書を指定します。
- 8 [保存] をクリックします。
- 9 左上隅の [インストール ダッシュボード] をクリックして、ダッシュボードに戻ります。
- 10 [Pivotal Application Service] をクリックします。
- 11 [設定] タブのナビゲーション ペインで、[ネットワーク] を選択します。
- 12 [コンテナ ネットワーク インターフェイスのプラグイン] で [外部] を選択します。
- 13 左上隅の [インストール ダッシュボード] をクリックして、ダッシュボードに戻ります。
- 14 [保存] をクリックします。
- 15 左上隅の [インストール ダッシュボード] をクリックして、ダッシュボードに戻ります。
- 16 [VMware NSX-T] をクリックします。
- 17 NSX Manager のアドレスを入力します。
- 18 NSX Manager の認証方法を選択します。

オプション	アクション
クライアント証明書認証	NSX Manager の証明書とプライベート キーを指定します。
ユーザー名とパスワードを使用した基本認証	NSX Manager 管理者のユーザー名とパスワードを指定します。

- 19 [NSX Manager CA 証明書] フィールドで、証明書を指定します。
- 20 [保存] をクリックします。
- 21 ナビゲーション ペインで [NCP] を選択します。
- 22 [PAS Foundation 名] を入力します。

この文字列は、NSX API で個々の PAS Foundation を一意に指定するものです。また、この文字列は、PAS Foundation 用に NCP が作成する NSX リソース名のプリフィックスとして使用されます。

- 23 [オーバーレイ トランSPORT ゾーン] を入力します。
- 24 [Tier-0 ルーター] を入力します。
- 25 [コンテナ ネットワークの IP アドレス ブロック] を 1 つ以上指定します。
  - a [追加] をクリックします。
  - b [IP アドレス ブロック名] を入力します。新規または既存の IP アドレス ブロックを入力できます。
  - c 新しい IP アドレス ブロックを入力する場合のみ、ブロックを 10.1.0.0/16 のような CIDR 形式で指定します。

- 26 コンテナ ネットワークのサブネット プリフィックスを指定します。
- 27 [コンテナ ネットワークの SNAT を有効にする] をクリックして、SNAT を有効にします。
- 28 [組織ネットワークに外部 (NAT) IP アドレスを提供するために使用される IP アドレス プール] を 1 つ以上指定します。
- a [追加] をクリックします。
  - b [IP アドレス プール名] を入力します。新規または既存の IP アドレス プールを入力できます。
  - c 新規 IP アドレス プールを入力する場合のみ、CIDR および IP アドレス範囲を入力して IP アドレスを指定します。
- 29 (オプション) [最上位のファイアウォール セクション マーカー] を入力します。
- 30 (オプション) [最下位のファイアウォール セクション マーカー] を入力します。
- 31 (オプション) 次のオプションを有効または無効にします。

オプション	デフォルト値
ドロップしたアプリケーショントラフィックをログに記録	無効。有効にすると、ファイアウォール ルールによってドロップしたトラフィックがログに記録されます。
NCP のデバッグ レベルのログ記録を有効にする	有効。

- 32 [保存] をクリックします。
- 33 (オプション) ナビゲーション ペインで [NSX Node Agent] を選択します。
- a [NSX Node Agent のデバッグ レベルのログ記録を有効にする] を選択して、デバッグ レベルでのログの記録を有効にします。
  - b [保存] をクリックします。
- 34 左上隅の [インストール ダッシュボード] をクリックして、ダッシュボードに戻ります。
- 35 [変更を適用] をクリックします。

# ロード バランシング

# 5

NSX-T ロード バランサは Kubernetes と連携します。

この章には、次のトピックが含まれています。

## ■ ロード バランシングの設定

## ロード バランシングの設定

ロード バランシングの設定には、Kubernetes の LoadBalancer サービスまたは Ingress リソースと、NCP レプリケーション コントローラの設定が含まれます。

LoadBalancer タイプの Kubernetes サービスを設定して、レイヤー 4 のロード バランサを作成できます。サービスには、事前設定した外部の IP アドレス ブロックの IP アドレスが割り当てられます。ロード バランサは、この IP アドレスとサービス ポートを通じて公開されます。LoadBalancer に定義されている `loadBalancerIP` を使用して、IP アドレス プールの名前または ID を指定できます。LoadBalancer サービスの IP アドレスは、この IP アドレス プールから割り当てられます。`loadBalancerIP` が空白の場合、IP アドレスは事前設定した外部 IP アドレス ブロックから割り当てられます。

Kubernetes のリソースを設定して、レイヤー 7 のロード バランサを作成できます。Ingress リソースには、設定した外部の IP アドレス ブロックの IP アドレスが割り当てられます。ロード バランサはこの IP アドレスで公開されます。注：LoadBalancer タイプの Kubernetes サービスは、Ingress リソースのバックエンドとしてはサポートされません。

---

**注：** LoadBalancer サービスまたは Ingress リソースに特定の IP アドレスを割り当てることはできません。LoadBalancer サービスまたは Ingress リソースの作成時に IP アドレスを指定しても、すべて無視されます。

---

NCP でロード バランシングを設定するには、`ncp_rc.yml` ファイルで次のように設定します。

- 1 `use_native_loadbalancer` を **True** に設定します。
- 2 (オプション) `pool_algorithm` を '**ROUND\_ROBIN**' または '**LEAST\_CONNECTION/IP\_HASH**' に設定します。デフォルトは **ROUND\_ROBIN** です。
- 3 (オプション) `service_size` を **SMALL**、**MEDIUM**、または **LARGE** に設定します。デフォルトは **SMALL** です。

**LEAST\_CONNECTION/IP\_HASH** アルゴリズムの場合、トラフィックの送信元 IP アドレスが同じであれば、同じバックエンド ポッドに送信されます。

小規模の NSX-T ロード バランサでは、次がサポートされます。

- NSX-T 仮想サーバ 10 台。
- NSX-T プール 10 個。
- NSX-T プール メンバー 30 個。
- LoadBalancer サービス用ポート 8 個。
- LoadBalancer サービスと Ingress リソース限定のポート合計 10 個。
- LoadBalancer サービスと Ingress リソースによって参照されるエンドポイント合計 30 台。

中規模の NSX-T ロード バランサでは、次がサポートされます。

- NSX-T 仮想サーバ 100 台。
- NSX-T プール 100 個。
- NSX-T プール メンバー 300 個。
- LoadBalancer サービス用ポート 98 個。
- LoadBalancer サービスと Ingress リソース限定のポート合計 100 個。
- LoadBalancer サービスと Ingress リソースによって参照されるエンドポイント合計 300 台。

大規模の NSX-T ロード バランサでは、次がサポートされます。

- NSX-T 仮想サーバ 1,000 台。
- NSX-T プール 1000 個。
- NSX-T プール メンバー 3,000 個。
- LoadBalancer サービス用ポート 998 個。
- LoadBalancer サービスと Ingress リソース限定のポート合計 1000 個。
- LoadBalancer サービスと Ingress リソースによって参照されるエンドポイント合計 3000 台。

ロード バランサが作成された後に、構成ファイルを更新してロード バランサのサイズを変更することはできません。変更するには、ユーザー インターフェイスまたは API を使用します。

## 入力方向 (Ingress)

- NSX-T では、TLS 仕様の Ingress と、TLS 仕様でない Ingress 用に、レイヤー 7 のロード バランサ仮想サーバが 1 つずつ作成されます。
- すべての Ingress が、単一の IP アドレスを取得します。
- TLS 仕様でない Ingress は、HTTP 仮想サーバ（ポート 80）上でホストされます。
- TLS 仕様の Ingress は、HTTPS 仮想サーバ（ポート 443）上でホストされます。ロード バランサは SSL サーバとして機能し、クライアントの SSL 接続を終了します。

- Secret と Ingress の作成順序は重要ではありません。Secret オブジェクトがある場合、それを参照する Ingress があると、証明書は NSX-T にインポートされます。Secret が削除された場合、または最後に Ingress が行った Secret 参照が削除された場合は、Secret に対応する証明書は削除されます。
- Ingress への TLS セクションの追加または削除はサポートされています。Ingress の仕様から `tls` キーが削除されると、Ingress ルールは HTTPS 仮想サーバ（ポート 443）から HTTP 仮想サーバ（ポート 80）に転送されます。同様に、Ingress の仕様に `tls` キーが追加されると、Ingress ルールは HTTP 仮想サーバ（ポート 80）から HTTPS 仮想サーバ（ポート 443）に転送されます。
- Ingress の定義で、単一クラスタに対するルールが重複している場合、最初のルールのみが適用されます。
- 1 つのクラスタでサポートされるのは、デフォルトのバックエンドを持つ単一の Ingress のみです。どの Ingress ルールとも一致しないトラフィックは、デフォルトのバックエンドに転送されます。
- デフォルトのバックエンドを持つ Ingress が複数ある場合は、最初の 1 つのみが設定されます。その他の Ingress は、エラーと見なされます。
- ワイルドカードによる URI マッチングでは、正規表現の文字「`.`」と「`*`」がサポートされます。たとえば、パス「`/coffee/*`」は、「`/coffee/`」の後に 0 個または 1 個以上の文字が続く文字列、たとえば「`/coffee/`」、「`/coffee/a`」、「`/coffee/b`」と一致します。「`/coffee`」、「`/coffeecup`」、「`/coffeecup/a`」などとは一致しません。パスに「`*`」が含まれている場合、たとえば、「`/tea/*`」は、「`/tea`」の後に 0 個または 1 個以上の文字が続く文字列、たとえば「`/tea`」、「`/tea/`」、「`/teacup`」、「`/teacup/`」、「`/tea/a`」、「`/teacup/b`」などとは一致することに注意してください。この場合、正規表現の特殊文字「`*`」もワイルドカード文字として機能します。

Ingress の仕様の例 :

```
kind: Ingress
metadata:
  name: cafe-ingress
spec:
  rules:
  - http:
      paths:
      - path: /tea/*           #Matches /tea, /tea/, /teacup, /teacup/, /tea/a, /teacup/b, etc.
        backend:
          serviceName: tea-svc
          servicePort: 80
      - path: /coffee/*      #Matches /coffee/, /coffee/a but NOT /coffee, /coffeecup, etc.
        backend:
          serviceName: coffee-svc
          servicePort: 80
```

- Ingress のリソースにアノテーションを追加して、HTTP URL リクエスト rewrite を設定できます。次はその例です。

```
kind: Ingress
metadata:
  name: cafe-ingress
  annotations:
    ncp/rewrite_target: "/"
spec:
  rules:
```

```

- host: cafe.example.com
  http:
    paths:
      - path: /tea
        backend:
          serviceName: tea-svc
          servicePort: 80
      - path: /coffee
        backend:
          serviceName: coffee-svc
          servicePort: 80

```

パス「/tea」および「/coffee」は、バックエンド サービスへの URL 送信前に「/」に書き換えられます。

## レイヤー 7 ロード バランサおよびネットワーク ポリシー

NSX ロード バランサ仮想サーバからポッドにトラフィックを転送する場合は、送信元 IP アドレスが Tier-1 ルーターのアップリンク ポートの IP アドレスになります。このアドレスはプライベート Tier-1 移行ネットワーク上にあるため、許可されるべきトラフィックが CIDR ベースのネットワーク ポリシーにより許可されないことがあります。この問題を避けるには、Tier-1 ルーターのアップリンク ポートの IP アドレスが、許可されている CIDR ブロックに含まれるようにネットワーク ポリシーを設定する必要があります。この内部 IP アドレスは、`status.loadbalancer.ingress.ip` フィールドの Ingress の仕様の一部および Ingress リソースのアノテーション (`ncp/internal_ip_for_policy`) としてユーザーに表示されます。

たとえば、仮想サーバの外部 IP アドレスが 4.4.0.5 であり、内部 Tier-1 ルーターのアップリンク ポートの IP アドレスが 100.64.224.11 の場合、Ingress の仕様は次のようになります。

```

kind: Ingress
...
status:
  loadBalancer:
    ingress:
      - ip: 4.4.0.5
      - ip: 100.64.224.11

```

Ingress リソースのアノテーションは次のようになります。

```
ncp/internal_ip_for_policy: 100.64.224.11
```

## LoadBalancer サービス

- NSX-T は、各サービス ポートにレイヤー 4 のロード バランサを作成します。
- TCP および UDP の両方がサポートされます。
- 各サービスには一意の IP アドレスが割り当てられます。

# NSX-T Container Plug-in の管理

# 6

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。

**注：** ESXi 6.5 で稼働中のコンテナ ホストの仮想マシンを、別の ESXi 6.5 ホストに vMotion で移行する場合、移行元のコンテナ ホスト上のコンテナから、移行先のコンテナ ホスト上のコンテナへの接続は失われます。この問題は、コンテナ ホストの vNIC を切断し、再度接続することで解決できます。この問題は、ESXi 6.5 Update 1 以降では発生しません。

Hyperbus では、ハイパーバイザーの VLAN ID 4093 と 4094 は PVLAN の設定用に予約されており、ID を変更することはできません。VLAN の競合を避けるため、同じ VLAN ID の VLAN 論理スイッチまたは VTEP vmknic は設定しないでください。

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
- エラー コード

## 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、CLI プロンプトを表示することもできます。

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

表 6-1. NCP コンテナの CLI コマンド

タイプ	コマンド	注
状態	<code>get ncp-master status</code>	Kubernetes と PCF の両方が対象。
状態	<code>get ncp-nsx status</code>	Kubernetes と PCF の両方が対象。
状態	<code>get ncp-watcher &lt;watcher-name&gt;</code>	Kubernetes と PCF の両方が対象。
状態	<code>get ncp-watchers</code>	Kubernetes と PCF の両方が対象。
状態	<code>get ncp-k8s-api-server status</code>	Kubernetes のみが対象。
状態	<code>check projects</code>	Kubernetes のみが対象。
状態	<code>check project &lt;project-name&gt;</code>	Kubernetes のみが対象。
状態	<code>get ncp-bbs status</code>	PCF のみが対象。
状態	<code>get ncp-capi status</code>	PCF のみが対象。
状態	<code>get ncp-policy-server status</code>	PCF のみが対象。

表 6-1. NCP コンテナの CLI コマンド (続き)

タイプ	コマンド	注
キャッシュ	get project-caches	Kubernetes のみが対象。
キャッシュ	get project-cache <project-name>	Kubernetes のみが対象。
キャッシュ	get namespace-caches	Kubernetes のみが対象。
キャッシュ	get namespace-cache <namespace-name>	Kubernetes のみが対象。
キャッシュ	get pod-caches	Kubernetes のみが対象。
キャッシュ	get pod-cache <pod-name>	Kubernetes のみが対象。
キャッシュ	get ingress-caches	Kubernetes のみが対象。
キャッシュ	get ingress-cache <ingress-name>	Kubernetes のみが対象。
キャッシュ	get ingress-controllers	Kubernetes のみが対象。
キャッシュ	get ingress-controller <ingress-controller-name>	Kubernetes のみが対象。
キャッシュ	get network-policy-caches	Kubernetes のみが対象。
キャッシュ	get network-policy-cache <pod-name>	Kubernetes のみが対象。
キャッシュ	get asg-caches	PCF のみが対象。
キャッシュ	get asg-cache <asg-ID>	PCF のみが対象。
キャッシュ	get org-caches	PCF のみが対象。
キャッシュ	get org-cache <org-ID>	PCF のみが対象。
キャッシュ	get space-caches	PCF のみが対象。
キャッシュ	get space-cache <space-ID>	PCF のみが対象。
キャッシュ	get app-caches	PCF のみが対象。
キャッシュ	get app-cache <app-ID>	PCF のみが対象。
キャッシュ	get instance-caches <app-ID>	PCF のみが対象。
キャッシュ	get instance-cache <app-ID> <instance-ID>	PCF のみが対象。
キャッシュ	get policy-caches	PCF のみが対象。
サポート	get ncp-log file <filename>	Kubernetes と PCF の両方が対象。
サポート	get ncp-log-level	Kubernetes と PCF の両方が対象。
サポート	set ncp-log-level <log-level>	Kubernetes と PCF の両方が対象。
サポート	get support-bundle file <filename>	Kubernetes のみが対象。
サポート	get node-agent-log file <filename>	Kubernetes のみが対象。
サポート	get node-agent-log file <filename> <node-name>	Kubernetes のみが対象。

表 6-2. NSX Node Agent コンテナの CLI コマンド

タイプ	コマンド
状態	get node-agent-hyperbus status
キャッシュ	get container-cache <container-name>
キャッシュ	get container-caches

表 6-3. NSX Kube-Proxy コンテナの CLI コマンド

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

## NCP コンテナの状態コマンド

- NCP マスターの状態を表示します。

```
get ncp-master status
```

例 :

```
kubenode> get ncp-master status
This instance is not the NCP master
Current NCP Master id is a4h83eh1-b8dd-4e74-c71c-cbb7cc9c4c1c
Last master update at Wed Oct 25 22:46:40 2017
```

- NCP と NSX Manager の間の接続の状態を表示します。

```
get ncp-nsx status
```

例 :

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

- Ingress、名前空間、ポッド、サービスの監視の状態を表示します。

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

例 :

```
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
```

```
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
```

- NCP と Kubernetes API サーバ間の接続の状態を表示します。

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

例 :

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

- すべてのプロジェクトまたは特定のプロジェクトを確認します。

```
check projects
check project <project-name>
```

例 :

```
kubecall> check projects
default:
  Tier-1 link port for router 1b90a61f-0f2c-4768-9eb6-ea8954b4f327 is missing
  Switch 40a6829d-c3aa-4e17-ae8a-7f7910fdf2c6 is missing

ns1:
  Router 8accc9cd-9883-45f6-81b3-0d1fb2583180 is missing

kubecall> check project default
Tier-1 link port for router 1b90a61f-0f2c-4768-9eb6-ea8954b4f327 is missing
Switch 40a6829d-c3aa-4e17-ae8a-7f7910fdf2c6 is missing
```

- NCP と PCF BBS 間の接続の状態を確認します。

```
get ncp-bbs status
```

例 :

```
node> get ncp-bbs status
BBS Server status: Healthy
```

- NCP と PCF CAPI 間の接続の状態を確認します。

```
get ncp-capi status
```

例 :

```
node> get ncp-capi status
CAPI Server status: Healthy
```

- NCP と PCF のポリシー サーバ間の接続の状態を確認します。

```
get ncp-policy-server status
```

例 :

```
node> get ncp-bbs status
Policy Server status: Healthy
```

## 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 :

```
kubenode> get project-caches
default:
  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:
  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

testns:
  ext_pool_id: 346a0f36-7b5a-4ecc-ad32-338dcb92316f
  labels:
    ns: myns
    project: myproject
  logical-router: 4dc8f8a9-69b4-4ff7-8fb7-d2625dc77efa
  logical-switch:
    id: 6111a99a-6e06-4faa-a131-649f10f7c815
    ip_pool_id: 51ca058d-c3dc-41fd-8f2d-e69006ab1b3d
    subnet: 50.0.2.0/24
    subnet_id: 34f79811-bd29-4048-a67d-67ceac97eb98
  project_nsgroup: 9606afee-6348-4780-9dbe-91abfd23e475
  snat_ip: 4.4.0.3

kubenode> get project-cache default
  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

kubenode> get namespace-caches
default:
```

```

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:
  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

testns:
  ext_pool_id: 346a0f36-7b5a-4ecc-ad32-338dcb92316f
  labels:
    ns: myns
    project: myproject
  logical-router: 4dc8f8a9-69b4-4ff7-8fb7-d2625dc77efa
  logical-switch:
    id: 6111a99a-6e06-4faa-a131-649f10f7c815
    ip_pool_id: 51ca058d-c3dc-41fd-8f2d-e69006ab1b3d
    subnet: 50.0.2.0/24
    subnet_id: 34f79811-bd29-4048-a67d-67ceac97eb98
  project_nsgroup: 9606afee-6348-4780-9dbe-91abfd23e475
  snat_ip: 4.4.0.3

```

```

kubenode> get namespace-cache default
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

```

#### ■ 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 :

```

kubenode> get pod-caches
nsx.default.nginx-rc-uq2lv:
  cif_id: 2af9f734-37b1-4072-ba88-abbf935bf3d4
  gateway_ip: 10.0.0.1
  host_vif: d6210773-5c07-4817-98db-451bd1f01937
  id: 1c8b5c52-3795-11e8-ab42-005056b198fb
  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

nsx.testns.web-pod-1:
  cif_id: ce134f21-6be5-43fe-afbf-aaca8c06b5cf
  gateway_ip: 50.0.2.1
  host_vif: d6210773-5c07-4817-98db-451bd1f01937
  id: 3180b521-270e-11e8-ab42-005056b198fb
  ingress_controller: False
  ip: 50.0.2.3/24
  labels:
    app: nginx-new
    role: db
    tier: cache
  mac: 02:50:56:00:20:02
  port_id: 81bc2b8e-d902-4cad-9fc1-aabdc32ecaf8
  vlan: 3

kubenode> get pod-cache nsx.default.nginx-rc-uq2lv
  cif_id: 2af9f734-37b1-4072-ba88-abbf935bf3d4
  gateway_ip: 10.0.0.1
  host_vif: d6210773-5c07-4817-98db-451bd1f01937
  id: 1c8b5c52-3795-11e8-ab42-005056b198fb
  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

```

- すべての Ingress のキャッシュまたは特定のキャッシュを取得します。

```

get ingress caches
get ingress-cache <ingress-name>

```

例 :

```

kubenode> get ingress-caches
nsx.default.cafe-ingress:
  ext_pool_id: cc02db70-539a-4934-a938-5b851b3e485b
  lb_virtual_server:
    id: 895c7f43-c56e-4b67-bb4c-09d68459d416
    lb_service_id: 659eefc6-33d1-4672-a419-344b877f528e
    name: dgo2-http
    type: http
  lb_virtual_server_ip: 5.5.0.2
  name: cafe-ingress
  rules:
    host: cafe.example.com
    http:

```

```

      paths:
        path: /coffee
        backend:
          serviceName: coffee-svc
          servicePort: 80
        lb_rule:
          id: 4bc16bdd-abd9-47fb-a09e-21e58b2131c3
          name: dgo2-default-cafe-ingress/coffee

kubensode> get ingress-cache nsx.default.cafe-ingress
ext_pool_id: cc02db70-539a-4934-a938-5b851b3e485b
lb_virtual_server:
  id: 895c7f43-c56e-4b67-bb4c-09d68459d416
  lb_service_id: 659eefc6-33d1-4672-a419-344b877f528e
  name: dgo2-http
  type: http
lb_virtual_server_ip: 5.5.0.2
name: cafe-ingress
rules:
  host: cafe.example.com
  http:
    paths:
      path: /coffee
      backend:
        serviceName: coffee-svc
        servicePort: 80
      lb_rule:
        id: 4bc16bdd-abd9-47fb-a09e-21e58b2131c3
        name: dgo2-default-cafe-ingress/coffee

```

- すべての Igress Controller、または特定の Igress Controller の情報を取得します。これには、無効なコントローラも含まれます。

```

get ingress controllers
get ingress-controller <ingress-controller-name>

```

例 :

```

kubensode> get ingress-controllers
native-load-balancer:
  ingress_virtual_server:
    http:
      default_backend_tags:
        id: 895c7f43-c56e-4b67-bb4c-09d68459d416
        pool_id: None
      https_terminated:
        default_backend_tags:
          id: 293282eb-f1a0-471c-9e48-ba28d9d89161
          pool_id: None
        lb_ip_pool_id: cc02db70-539a-4934-a938-5b851b3e485b
      loadbalancer_service:
        first_avail_index: 0

```

```

    lb_services:
      id: 659eefc6-33d1-4672-a419-344b877f528e
      name: dgo2-bfmxi
      t1_link_port_ip: 100.64.128.5
      t1_router_id: cb50deb2-4460-45f2-879a-1b94592ae886
      virtual_servers:
        293282eb-f1a0-471c-9e48-ba28d9d89161
        895c7f43-c56e-4b67-bb4c-09d68459d416
    ssl:
      ssl_client_profile_id: aff205bb-4db8-5a72-8d67-218cdc54d27b
    vip: 5.5.0.2

nsx.default.nginx-ingress-rc-host-ed3og
  ip: 10.192.162.201
  mode: hostnetwork
  pool_id: 5813c609-5d3a-4438-b9c3-ea3cd6de52c3

kubenode> get ingress-controller native-load-balancer
  ingress_virtual_server:
    http:
      default_backend_tags:
        id: 895c7f43-c56e-4b67-bb4c-09d68459d416
        pool_id: None
      https_terminated:
        default_backend_tags:
          id: 293282eb-f1a0-471c-9e48-ba28d9d89161
          pool_id: None
    lb_ip_pool_id: cc02db70-539a-4934-a938-5b851b3e485b
    loadbalancer_service:
      first_avail_index: 0
      lb_services:
        id: 659eefc6-33d1-4672-a419-344b877f528e
        name: dgo2-bfmxi
        t1_link_port_ip: 100.64.128.5
        t1_router_id: cb50deb2-4460-45f2-879a-1b94592ae886
        virtual_servers:
          293282eb-f1a0-471c-9e48-ba28d9d89161
          895c7f43-c56e-4b67-bb4c-09d68459d416
      ssl:
        ssl_client_profile_id: aff205bb-4db8-5a72-8d67-218cdc54d27b
    vip: 5.5.0.2

```

- ネットワーク ポリシー キャッシュまたは特定のポリシー キャッシュを取得します。

```

get network-policy caches
get network-policy-cache <network-policy-name>

```

例 :

```

kubenode> get network-policy-caches
  nsx.testns.allow-tcp-80:
    dest_labels: None
    dest_pods:

```

```

    50.0.2.3
  match_expressions:
    key: tier
    operator: In
    values:
      cache
  name: allow-tcp-80
  np_dest_ip_set_ids:
    22f82d76-004f-4d12-9504-ce1cb9c8aa00
  np_except_ip_set_ids:
  np_ip_set_ids:
    14f7f825-f1a0-408f-bbd9-bb2f75d44666
  np_isol_section_id: c8d93597-9066-42e3-991c-c550c46b2270
  np_section_id: 04693136-7925-44f2-8616-d809d02cd2a9
  ns_name: testns
  src_egress_rules: None
  src_egress_rules_hash: 97d170e1550eee4afc0af065b78cda302a97674c
  src_pods:
    50.0.2.0/24
  src_rules:
    from:
      namespaceSelector:
        matchExpressions:
          key: tier
          operator: DoesNotExist
        matchLabels:
          ns: myns
    ports:
      port: 80
      protocol: TCP
  src_rules_hash: e4ea7b8d91c1e722670a59f971f8fcc1a5ac51f1

```

```

kubenode> get network-policy-cache nsx.testns.allow-tcp-80
dest_labels: None
dest_pods:
  50.0.2.3
match_expressions:
  key: tier
  operator: In
  values:
    cache
name: allow-tcp-80
np_dest_ip_set_ids:
  22f82d76-004f-4d12-9504-ce1cb9c8aa00
np_except_ip_set_ids:
np_ip_set_ids:
  14f7f825-f1a0-408f-bbd9-bb2f75d44666
np_isol_section_id: c8d93597-9066-42e3-991c-c550c46b2270
np_section_id: 04693136-7925-44f2-8616-d809d02cd2a9
ns_name: testns
src_egress_rules: None
src_egress_rules_hash: 97d170e1550eee4afc0af065b78cda302a97674c
src_pods:
  50.0.2.0/24

```

```
src_rules:
  from:
    namespaceSelector:
      matchExpressions:
        key: tier
        operator: DoesNotExist
      matchLabels:
        ns: myns
    ports:
      port: 80
      protocol: TCP
src_rules_hash: e4ea7b8d91c1e722670a59f971f8fcc1a5ac51f1
```

- すべての ASG キャッシュまたは特定の ASG キャッシュを取得します。

```
get asg-caches
get asg-cache <asg-ID>
```

例 :

```
node> get asg-caches
edc04715-d04c-4e63-abbcd-b601a668db6:
  fws_id: 3c66f40a-5378-46d7-a7e2-bee4ba72a4cc
  name: org-85_tcp_80_asg
  rules:
    destinations:
      66.10.10.0/24
    ports:
      80
    protocol: tcp
    rule_id: 4359
  running_default: False
  running_spaces:
    75bc164d-1214-46f9-80bb-456a8fbccbfd
  staging_default: False
  staging_spaces:

node> get asg-cache edc04715-d04c-4e63-abbcd-b601a668db6
fws_id: 3c66f40a-5378-46d7-a7e2-bee4ba72a4cc
name: org-85_tcp_80_asg
rules:
  destinations:
    66.10.10.0/24
  ports:
    80
  protocol: tcp
  rule_id: 4359
running_default: False
running_spaces:
  75bc164d-1214-46f9-80bb-456a8fbccbfd
staging_default: False
staging_spaces:
```

- すべての組織キャッシュまたは特定の組織キャッシュを取得します。

```
get org-caches
get org-cache <org-ID>
```

例 :

```
node> get org-caches
ebb8b4f9-a40f-4122-bf21-65c40f575aca:
  ext_pool_id: 9208a8b8-57d7-4582-9c1f-7a7cefa104f5
  isolation:
    isolation_section_id: d6e7ff95-4737-4e34-91d4-27601897353f
  logical-router: 94a414a2-551e-4444-bae6-3d79901a165f
  logical-switch:
    id: d74807e8-8f74-4575-b26b-87d4fdbafd3c
    ip_pool_id: 1b60f16f-4a30-4a3d-93cc-bfb08a5e3e02
    subnet: 50.0.48.0/24
    subnet_id: a458d3aa-bea9-4684-9957-d0ce80d11788
  name: org-50
  snat_ip: 70.0.0.49
  spaces:
    e8ab7aa0-d4e3-4458-a896-f33177557851

node> get org-cache ebb8b4f9-a40f-4122-bf21-65c40f575aca
  ext_pool_id: 9208a8b8-57d7-4582-9c1f-7a7cefa104f5
  isolation:
    isolation_section_id: d6e7ff95-4737-4e34-91d4-27601897353f
  logical-router: 94a414a2-551e-4444-bae6-3d79901a165f
  logical-switch:
    id: d74807e8-8f74-4575-b26b-87d4fdbafd3c
    ip_pool_id: 1b60f16f-4a30-4a3d-93cc-bfb08a5e3e02
    subnet: 50.0.48.0/24
    subnet_id: a458d3aa-bea9-4684-9957-d0ce80d11788
  name: org-50
  snat_ip: 70.0.0.49
  spaces:
    e8ab7aa0-d4e3-4458-a896-f33177557851
```

- すべての容量キャッシュまたは特定の容量キャッシュを取得します。

```
get space-caches
get space-cache <space-ID>
```

例 :

```
node> get space-caches
global_security_group:
  name: global_security_group
  running_nsgroup: 226d4292-47fb-4c2e-a118-449818d8fa98
  staging_nsgroup: 7ebbf7f5-38c9-43a3-9292-682056722836

7870d134-7997-4373-b665-b6a910413c47:
  name: test-space1
```

```

org_id: a8423bc0-4b2b-49fb-bbff-a4badf21eb09
running_nsgroup: 4a3d9bcc-be36-47ae-bff8-96448fecf307
running_security_groups:
    aa0c7c3f-a478-4d45-8afa-df5d5d7dc512
staging_security_groups:
    aa0c7c3f-a478-4d45-8afa-df5d5d7dc512

```

```

node> get space-cache 7870d134-7997-4373-b665-b6a910413c47
name: test-space1
org_id: a8423bc0-4b2b-49fb-bbff-a4badf21eb09
running_nsgroup: 4a3d9bcc-be36-47ae-bff8-96448fecf307
running_security_groups:
    aa0c7c3f-a478-4d45-8afa-df5d5d7dc512
staging_security_groups:
    aa0c7c3f-a478-4d45-8afa-df5d5d7dc512

```

- すべてのアプリケーションのキャッシュまたは特定のアプリケーションのキャッシュを取得します。

```

get app-caches
get app-cache <app-ID>

```

例 :

```

node> get app-caches
aff2b12b-b425-4d9f-b8e6-b6308644efa8:
  instances:
    b72199cc-e1ab-49bf-506d-478d:
      app_id: aff2b12b-b425-4d9f-b8e6-b6308644efa8
      cell_id: 0dda88bc-640b-44e7-8cea-20e83e873544
      cif_id: 158a1d7e-6ccc-4027-a773-55bb2618f51b
      gateway_ip: 192.168.5.1
      host_vif: 53475dfd-03e4-4bc6-b8ba-3d803725cbab
      id: b72199cc-e1ab-49bf-506d-478d
      index: 0
      ip: 192.168.5.4/24
      last_updated_time: 1522965828.45
      mac: 02:50:56:00:60:02
      port_id: a7c6f6bb-c472-4239-a030-bce615d5063e
      state: RUNNING
      vlan: 3
      name: hello2
      rg_id: a8423bc0-4b2b-49fb-bbff-a4badf21eb09
      space_id: 7870d134-7997-4373-b665-b6a910413c47

```

```

node> get app-cache aff2b12b-b425-4d9f-b8e6-b6308644efa8
instances:
  b72199cc-e1ab-49bf-506d-478d:
    app_id: aff2b12b-b425-4d9f-b8e6-b6308644efa8
    cell_id: 0dda88bc-640b-44e7-8cea-20e83e873544
    cif_id: 158a1d7e-6ccc-4027-a773-55bb2618f51b
    gateway_ip: 192.168.5.1
    host_vif: 53475dfd-03e4-4bc6-b8ba-3d803725cbab

```

```

id: b72199cc-e1ab-49bf-506d-478d
index: 0
ip: 192.168.5.4/24
last_updated_time: 1522965828.45
mac: 02:50:56:00:60:02
port_id: a7c6f6bb-c472-4239-a030-bce615d5063e
state: RUNNING
vlan: 3
name: hello2
org_id: a8423bc0-4b2b-49fb-bbfb-a4badf21eb09
space_id: 7870d134-7997-4373-b665-b6a910413c47

```

- アプリケーションのすべてのインスタンス キャッシュまたは特定のインスタンス キャッシュを取得します。

```

get instance-caches <app-ID>
get instance-cache <app-ID> <instance-ID>

```

例 :

```

node> get instance-caches aff2b12b-b425-4d9f-b8e6-b6308644efa8
b72199cc-e1ab-49bf-506d-478d:
  app_id: aff2b12b-b425-4d9f-b8e6-b6308644efa8
  cell_id: 0dda88bc-640b-44e7-8cea-20e83e873544
  cif_id: 158a1d7e-6ccc-4027-a773-55bb2618f51b
  gateway_ip: 192.168.5.1
  host_vif: 53475dfd-03e4-4bc6-b8ba-3d803725cbab
  id: b72199cc-e1ab-49bf-506d-478d
  index: 0
  ip: 192.168.5.4/24
  last_updated_time: 1522965828.45
  mac: 02:50:56:00:60:02
  port_id: a7c6f6bb-c472-4239-a030-bce615d5063e
  state: RUNNING
  vlan: 3

node> get instance-cache aff2b12b-b425-4d9f-b8e6-b6308644efa8 b72199cc-e1ab-49bf-506d-478d
  app_id: aff2b12b-b425-4d9f-b8e6-b6308644efa8
  cell_id: 0dda88bc-640b-44e7-8cea-20e83e873544
  cif_id: 158a1d7e-6ccc-4027-a773-55bb2618f51b
  gateway_ip: 192.168.5.1
  host_vif: 53475dfd-03e4-4bc6-b8ba-3d803725cbab
  id: b72199cc-e1ab-49bf-506d-478d
  index: 0
  ip: 192.168.5.4/24
  last_updated_time: 1522965828.45
  mac: 02:50:56:00:60:02
  port_id: a7c6f6bb-c472-4239-a030-bce615d5063e
  state: RUNNING
  vlan: 3

```

- すべてのポリシー キャッシュを取得します。

```

get policy-caches

```

例 :

```
node> get policy-caches
aff2b12b-b425-4d9f-b8e6-b6308644efa8:
  fws_id: 3fe27725-f139-479a-b83b-8576c9aedbef
  nsg_id: 30583a27-9b56-49c1-a534-4040f91cc333
  rules:
    8272:
      dst_app_id: aff2b12b-b425-4d9f-b8e6-b6308644efa8
      ports: 8382
      protocol: tcp
      src_app_id: f582ec4d-3a13-440a-afbd-97b7bfae21d1

f582ec4d-3a13-440a-afbd-97b7bfae21d1:
  nsg_id: d24b9f77-e2e0-4fba-b258-893223683aa6
  rules:
    8272:
      dst_app_id: aff2b12b-b425-4d9f-b8e6-b6308644efa8
      ports: 8382
      protocol: tcp
      src_app_id: f582ec4d-3a13-440a-afbd-97b7bfae21d1
```

## 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 バンドルには、**tier:nsx-networking** というラベルが付いており、ポッド内のすべてのコンテナのログファイルが含まれます。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 の `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 の `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
```

- Node Agent 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 Node Agent 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 の `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
```

- ログ レベルを取得して設定します。

使用可能なログ レベルは NOTSET、DEBUG、INFO、WARNING、ERROR、および CRITICAL です。

```
get ncp-log-level
set ncp-log-level <log level>
```

例 :

```
kubenode>get ncp-log-level
NCP log level is INFO

kubenode>set ncp-log-level DEBUG
NCP log level is changed to DEBUG
```

## NSX Node Agent コンテナの状態コマンド

- このノードの Node Agent と HyperBus 間の接続の状態を表示します。

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

例 :

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

## NSX Node Agent コンテナのキャッシュ コマンド

- NSX Node Agent コンテナの内部キャッシュを取得します。

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

例 :

```
kubenode> get container-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
```

```
kubenode> get container-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
```

## NSX Kube-Proxy コンテナの状態コマンド

- Kube プロキシと Kubernetes API サーバ間の接続の状態を表示します。

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

例 :

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

- Kube プロキシの監視の状態を表示します。

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

例 :

```
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
```

```
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
```

## ■ 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
```

## エラー コード

このセクションでは、さまざまなコンポーネントによって生成されるエラー コードについて説明します。

### NCP (NSX Container Plug-in) エラー コード

エラー コード	説明
NCP00001	設定が無効です
NCP00002	初期化に失敗しました
NCP00003	状態が無効です
NCP00004	アダプタが無効です
NCP00005	証明書が見つかりません
NCP00006	トークンが見つかりません
NCP00007	NSX の設定が無効です
NCP00008	NSX のタグが無効です
NCP00009	NSX の接続に失敗しました
NCP00010	ノード タグが見つかりません

エラー コード	説明
NCP00011	ノードの論理スイッチ ポートが無効です
NCP00012	親 VIF の更新に失敗しました
NCP00013	VLAN が不足しています
NCP00014	VLAN の解放に失敗しました
NCP00015	IP アドレス プールが不足しています
NCP00016	IP アドレスの解放に失敗しました
NCP00017	IP アドレス ブロックが不足しています
NCP00018	IP サブネットの作成に失敗しました
NCP00019	IP サブネットの削除に失敗しました
NCP00020	IP アドレス プールの作成に失敗しました
NCP00021	IP アドレス プールの削除に失敗しました
NCP00022	論理ルーターの作成に失敗しました
NCP00023	論理ルーターの更新に失敗しました
NCP00024	論理ルーターの削除に失敗しました
NCP00025	論理スイッチの作成に失敗しました

エラー コード	説明
NCP00026	論理スイッチの更新に失敗しました
NCP00027	論理スイッチの削除に失敗しました
NCP00028	論理ルーター ポートの作成に失敗しました
NCP00029	論理ルーター ポートの削除に失敗しました
NCP00030	論理スイッチ ポートの作成に失敗しました
NCP00031	論理スイッチ ポートの更新に失敗しました
NCP00032	論理スイッチ ポートの削除に失敗しました
NCP00033	ネットワーク ポリシーが見つかりません
NCP00034	ファイアウォールの作成に失敗しました
NCP00035	ファイアウォールの読み取りに失敗しました
NCP00036	ファイアウォールの更新に失敗しました
NCP00037	ファイアウォールの削除に失敗しました
NCP00038	複数のファイアウォールが見つかりました
NCP00039	NSGroup の作成に失敗しました
NCP00040	NSGroup の削除に失敗しました
NCP00041	IP セットの作成に失敗しました
NCP00042	IP セットの更新に失敗しました
NCP00043	IP セットの削除に失敗しました

エラー コード	説明
NCP00044	SNAT ルールの作成に失敗しました
NCP00045	SNAT ルールの削除に失敗しました
NCP00046	アダプタ API による接続に失敗しました
NCP00047	アダプタ ウォッチャの例外です
NCP00048	ロード バランサ サービスの削除に失敗しました
NCP00049	ロード バランサ仮想サーバの作成に失敗しました
NCP00050	ロード バランサ仮想サーバの更新に失敗しました
エラー コード	説明
NCP00051	ロード バランサ仮想サーバの削除に失敗しました
NCP00052	ロード バランサ プールの作成に失敗しました
NCP00053	ロード バランサ プールの更新に失敗しました
NCP00054	ロード バランサ プールの削除に失敗しました
NCP00055	ロード バランサ ルールの作成に失敗しました
NCP00056	ロード バランサ ルールの更新に失敗しました
NCP00057	ロード バランサ ルールの削除に失敗しました
NCP00058	ロード バランサ プールの IP アドレスの解放に失敗しました
NCP00059	ロード バランサ仮想サーバとサービスの関連付けが見つかりません
NCP00060	NSGroup の更新に失敗しました
NCP00061	ファイアウォール ルールの取得に失敗しました
NCP00062	NSGroup の条件がありません
NCP00063	ノード仮想マシンが見つかりません
NCP00064	ノード VIF が見つかりません
NCP00065	証明書のインポートに失敗しました
NCP00066	証明書のアンインポートに失敗しました
NCP00067	SSL バインドの更新に失敗しました
NCP00068	SSL プロファイルが見つかりません
NCP00069	IP アドレス プールが見つかりません
NCP00070	T0 Edge クラスタが見つかりません
NCP00071	IP アドレス プールの更新に失敗しました
NCP00072	ディスパッチャに失敗しました
NCP00073	NAT ルールの削除に失敗しました
NCP00074	論理ルーター ポートの取得に失敗しました
NCP00075	NSX の設定の確認に失敗しました

エラー コード	説明
NCP00076	SNAT ルールの更新に失敗しました
NCP00077	SNAT ルールが重複しています
NCP00078	ロード バランサのエンドポイントの追加に失敗しました
NCP00079	ロード バランサのエンドポイントの更新に失敗しました
NCP00080	ロード バランサ ルール プールの作成に失敗しました
NCP00081	ロード バランサ仮想サーバが見つかりません
NCP00082	IP セットの読み取りに失敗しました
NCP00083	SNAT プールの取得に失敗しました
NCP00084	ロード バランサ サービスの作成に失敗しました
NCP00085	ロード バランサ サービスの更新に失敗しました
NCP00086	論理ルーター ポートの更新に失敗しました
NCP00087	ロード バランサの初期化に失敗しました
NCP00088	IP アドレス プールが一意ではありません
NCP00089	レイヤー 7 ロード バランサ キャッシュの同期エラー
NCP00090	ロード バランサ プールが配置されていません
NCP00091	ロード バランサ ルール キャッシュの初期化エラー
NCP00092	SNAT プロセスが失敗しました
NCP00093	ロード バランサのデフォルト証明書エラー
NCP00094	ロード バランサのエンドポイントの削除に失敗しました
NCP00095	プロジェクトが見つかりません

## NSX ノード エージェントのエラー コード

エラー コード	説明
NCP01001	OVS アップリンクが見つかりません
NCP01002	ホストの MAC アドレスが見つかりません
NCP01003	OVS ポートの作成に失敗しました
NCP01004	ポッドが設定されていません
NCP01005	ポッドの設定に失敗しました
NCP01006	ポッドの設定解除に失敗しました
NCP01007	CNI ソケットが見つかりません
NCP01008	CNI の接続に失敗しました
NCP01009	CNI バージョンが一致しません
NCP01010	CNI メッセージの受信に失敗しました
NCP01011	CNI メッセージの転送に失敗しました
NCP01012	Hyperbus の接続に失敗しました

エラー コード	説明
NCP01013	Hyperbus バージョンが一致しません
NCP01014	Hyperbus メッセージの受信に失敗しました
NCP01015	Hyperbus メッセージの転送に失敗しました
NCP01016	GARP の送信に失敗しました
NCP01017	インターフェイスの設定に失敗しました

## nsx-kube-proxy のエラー コード

エラー コード	説明
NCP02001	無効なプロキシのゲートウェイ ポートです
NCP02002	プロキシ コマンドの実行に失敗しました
NCP02003	プロキシの確認に失敗しました

## CLI のエラー コード

エラー コード	説明
NCP03001	CLI の起動に失敗しました
NCP03002	CLI ソケットの作成に失敗しました
NCP03003	CLI ソケットの例外です
NCP03004	CLI クライアントの要求が無効です
NCP03005	CLI サーバの転送に失敗しました
NCP03006	CLI サーバの受信に失敗しました
NCP03007	CLI コマンドの実行に失敗しました

## Kubernetes のエラー コード

エラー コード	説明
NCP05001	Kubernetes の接続に失敗しました
NCP05002	Kubernetes の設定が無効です
NCP05003	Kubernetes の要求に失敗しました
NCP05004	Kubernetes のキーが見つかりません
NCP05005	Kubernetes のタイプが見つかりません
NCP05006	Kubernetes ウォッチャの例外です
NCP05007	Kubernetes リソースの長さが無効です
NCP05008	Kubernetes リソースのタイプが無効です
NCP05009	Kubernetes リソースの処理に失敗しました
NCP05010	Kubernetes サービスの処理に失敗しました

エラー コード	説明
NCP05011	Kubernetes エンドポイントの処理に失敗しました
NCP05012	Kubernetes Ingress ハンドルの処理に失敗しました
NCP05013	Kubernetes ネットワーク ポリシーの処理に失敗しました
NCP05014	Kubernetes ノードの処理に失敗しました
NCP05015	Kubernetes 名前空間の処理に失敗しました
NCP05016	Kubernetes ポッドの処理に失敗しました
NCP05017	Kubernetes Secret の処理に失敗しました
NCP05018	Kubernetes デフォルト バックエンドが失敗しました
NCP05019	一致式が Kubernetes でサポートされていません
NCP05020	Kubernetes のステータスの更新に失敗しました
NCP05021	Kubernetes のアノテーションの更新に失敗しました
NCP05022	Kubernetes の名前空間のキャッシュが見つかりません
NCP05023	Kubernetes Secret が見つかりません

## Pivotal Cloud Foundry のエラー コード

エラー コード	説明
NCP06001	PCF BBS の接続に失敗しました
NCP06002	PCF CAPI の接続に失敗しました
NCP06006	PCF キャッシュが見つかりません
NCP06007	PCF の不明なドメインです
NCP06020	PCF ポリシー サーバの接続に失敗しました
NCP06021	PCF ポリシーの処理に失敗しました
NCP06030	PCF イベントの処理に失敗しました
NCP06031	予期していない PCF のイベント タイプです
NCP06032	予期していない PCF のイベント インスタンスです
NCP06033	PCF タスクの削除に失敗しました