

NSX-T Container Plug-in for OpenShift : 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®

VMware Web サイトで最新の技術ドキュメントをご確認いただけます。

<https://docs.vmware.com/jp/>

VMware の Web サイトでは、最新の製品アップデートを提供しています。

本書に関するご意見、ご要望をお寄せください。フィードバック送信先：

docfeedback@vmware.com

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

ヴィ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

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

内容

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

1 NSX-T Container Plug-in の概要 5

互換性の要件 6

インストールの概要 6

2 NSX-T リソースのセットアップ 7

NSX-T リソースの設定 7

Tier-0 論理ルーターの作成と設定 10

3 NSX-T Container Plug-in と OpenShift のセットアップ 11

OpenShift 仮想マシンの展開 11

Ansible の hosts ファイルの準備 11

単一の Playbook を使用した NCP と OpenShift のインストール 13

CNI プラグイン、OVS、NCP Docker イメージのインストール 14

OpenShift Container Platform のインストール 16

NCP と NSX Node Agent の実行 16

セットアップに関する注意事項 18

4 ロード バランシング 21

ロード バランシングの設定 21

5 NSX-T Container Plug-in の管理 27

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 27

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 28

CIF 接続論理ポート 28

CLI コマンド 29

エラー コード 40

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

このガイドでは、NSX-T Container Plug-in (NCP) をインストールして管理する方法を説明します。また、NSX-T と OpenShift を統合する方法についても説明します。

対象読者

このガイドは、システム管理者とネットワーク管理者を対象としています。NSX-T および OpenShift のインストールと管理について理解していることが前提となります。

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

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

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

NSX CNI プラグインは、各 OpenShift 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。

- OpenShift クラスタに NSX-T 論理トポロジを自動的に作成し、OpenShift の名前空間に個別に論理ネットワークを作成します。
- OpenShift 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) をサポートしているため、OpenShift の名前空間に個別の SNAT IP を割り当てることができます。

注: NAT を設定するとき、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 OpenShift 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。
- NSX-T レイヤー 7 ロード バランサを使用する OpenShift Route を実装。
 - TLS Edge ターミネーションを含む HTTP Route と HTTPS Route をサポート。
 - 代替バックエンドとワイルドカード サブドメインを含むルートをサポート。
- NSX-T 論理スイッチ ポート上に、名前空間、ポッド名、ポッドのラベル用のタグを作成し、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 OpenShift クラスタをサポートします。

この章には、次のトピックが含まれています。

- [互換性の要件](#)
- [インストールの概要](#)

互換性の要件

NSX-T Container Plug-in (NCP) には、次の互換性の要件があります。

ソフトウェア製品	バージョン
NSX-T	2.1、2.2
コンテナ ホスト仮想マシン向けハイパーバイザー	<ul style="list-style-type: none"> ■ サポート対象の vSphere バージョン ■ RHEL KVM 7.4、7.5
コンテナ ホストのオペレーティング システム	RHEL 7.4、7.5
Platform as a Service	OpenShift 3.7、3.9
ゲスト vSwitch	OVS 2.8.1 (NSX-T 2.1 に同梱)、2.9.1 (NSX-T 2.2 に同梱)

インストールの概要

以下の手順に従って、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 OpenShift 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、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 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 OpenShift 仮想マシンを展開します。
- 8 Ansible hosts ファイルを準備します。
- 9 (オプション 1) 単一の Playbook を使用して NCP と OpenShift をインストールします。
(オプション 2) CNI プラグイン、OVS (Open vSwitch)、NCP Docker イメージをインストールします。次に、OpenShift Container Platform をインストールします。
- 10 NCP と NSX Node Agent を実行します。

提供されている Playbook を使用して NCP をインストールする場合、手順 2～6 は必要ありません。[「単一の Playbook を使用した NCP と OpenShift のインストール」](#) および [「CNI プラグイン、OVS、NCP Docker イメージのインストール」](#) を参照してください。

NSX-T リソースのセットアップ

OpenShift ノードにネットワーク機能を提供するには、NSX-T リソースを作成する必要があります。これらのリソースは、NSX Manager のグラフィカル ユーザー インターフェイス (GUI) 使用して手動で設定します。または、Ansible Playbook を使用して設定プロセスを自動化することも可能です。

このセクションでは、NSX-T リソースを手動で作成する方法について説明します。プロセスを自動化するには、[「CNI プラグイン、OVS、NCP Docker イメージのインストール」](#) を参照してください。

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。

SNAT 以外の名前空間に IP アドレス ブロックを作成することもできます。

`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。

SNAT の IP アドレス プール

SNAT ルールによってポッドの IP アドレスを変換する場合、または SNAT/DNAT ルールによって Ingress Controller を使用する場合、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 アドレスともいいます。

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 (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名前空間を指定できます。

- スコープ : **ncp/owner**、タグ : **ns:<namespace_UUID>**

名前空間 UUID は、次のいずれかのコマンドで取得できます。

```
oc get ns -o yaml
```

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の IP アドレスが割り当てられた後にタグを変更すると、サービスの SNAT 設定が変更されるか NCP が再起動されるまで、割り当て済みの 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、プールから IP アドレスをすべてのサービスに割り当てることができます。

(オプション) ファイアウォールのマーカー セクション

管理者が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 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 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。

NSX-T Container Plug-in と OpenShift のセットアップ

3

この章では、NSX-T Container Plug-in (NCP) と OpenShift のインストールと設定方法について説明します。

この章には、次のトピックが含まれています。

- [OpenShift 仮想マシンの展開](#)
- [Ansible の hosts ファイルの準備](#)
- [単一の Playbook を使用した NCP と OpenShift のインストール](#)
- [CNI プラグイン、OVS、NCP Docker イメージのインストール](#)
- [OpenShift Container Platform のインストール](#)
- [NCP と NSX Node Agent の実行](#)
- [セットアップに関する注意事項](#)

OpenShift 仮想マシンの展開

NSX-T Container Plug-in をインストールする前に、OpenShift をインストールする必要があります。1 つ以上のマスタを展開する必要があります。

詳細については、<https://docs.openshift.com> を参照してください。

次のステップ

Ansible hosts ファイルを準備します。「[\[「Ansible の hosts ファイルの準備」\]](#)」を参照してください。

Ansible の hosts ファイルの準備

Ansible の **hosts** ファイルには、OpenShift クラスタのノードが定義されます。

手順

- 1 <https://github.com/vmware/nsx-integration-for-openshift> で、NCP GitHub リポジトリのクローンを作成します。**hosts** ファイルは、**openshift-ansible-nsx** ディレクトリにあります。**hosts** ファイルは **openshift-ansible-nsx** ディレクトリ内に保存する必要があります。一部の Playbook は、このパスが **hosts** ファイルのパスであることを前提としています。

- 2 [masters] と [nodes] セクションに、OpenShift 仮想マシンのホスト名と IP アドレスを指定します。次はその例です。

```
[masters]
admin.rhel.osmaster ansible_ssh_host=101.101.101.4

[single_master]
admin.rhel.osmaster ansible_ssh_host=101.101.101.4

[nodes]
admin.rhel.osmaster ansible_ssh_host=101.101.101.4 openshift_ip=101.101.101.4
openshift_schedulable=true openshift_hostname=admin.rhel.osmaster
admin.rhel.osnode ansible_ssh_host=101.101.101.5 openshift_ip=101.101.101.5
openshift_hostname=admin.rhel.osnode

[etcd]

[OSEv3:children]
masters
nodes
etcd
```

openshift_ip は、クラスタの内部 IP アドレスを定義します。使用するインターフェイスがデフォルトでない場合には設定が必要になります。**single_master** 変数は、マスター ノードの ncp 関連のロールで使用され、NSX-T 管理プレーンのリソース設定など、特定のタスクを 1 回のみ実行します。

- 3 SSH アクセス権を設定して、Ansible ロールが実行されているノード（通常はマスター ノード）から、パスワードを使用せずにすべてのノードにアクセスできるようにします。

```
ssh-keygen
ssh-copy-id -i ~/.ssh/id_rsa.pub root@admin.rhel.osnode
```

- 4 [OSEv3:vars] セクションを更新します。すべてのパラメータの詳細については、OpenShift Container Platform のドキュメントで高度なインストールの説明を参照してください (<https://docs.openshift.com> で「高度なインストール」を検索)。次はその例です。

```
# Set the default route fqdn
openshift_master_default_subdomain=apps.corp.local

os_sdn_network_plugin_name=cni
openshift_use_openshift_sdn=false
openshift_node_sdn_mtu=1500

# If ansible_ssh_user is not root, ansible_become must be set to true
ansible_become=true

openshift_master_default_subdomain
    This is the default subdomain used in the OpenShift routes for External LB

os_sdn_network_plugin_name
    Set to 'cni' for the NSX Integration
```

```

openshift_use_openshift_sdn
    Set to false to disable the built-in OpenShift SDN solution

openshift_hosted_manage_router
    Set to false to disable creation of router during installation. The router has to
    be manually started after NCP and nsx-node-agent are running.

openshift_hosted_manage_registry
    Set to false to disable creation of registry during installation. The registry has
    to be manually started after NCP and nsx-node-agent are running.

deployment_type
    Set to openshift-enterprise

openshift_hosted_manage_registry
    Set to false to disable auto creation of registry

openshift_hosted_manage_router
    Set to false to disable auto creation of router

openshift_enable_service_catalog
    Set to false to disable service_catalog

(For OpenShift 3.9 only) skip_sanity_checks
    Set to true

(For OpenShift 3.9 only) openshift_web_console_install
    Set to false

```

5 すべてのホストに接続できることを確認します。

```
ansible OSEv3 -i /PATH/TO/HOSTS/hosts -m ping
```

結果は次のようになります。接続できない場合には、接続の問題を解決します。

```

openshift-node1 | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
openshift-master | SUCCESS => {
  "changed": false,
  "ping": "pong"
}

```

次のステップ

CNI プラグインと OVS をインストールします。「[\[CNI プラグイン、OVS、NCP Docker イメージのインストール\]](#)」を参照してください。

単一の Playbook を使用した NCP と OpenShift のインストール

NCP と OpenShift のインストールには、単一の Playbook を使用するか、個別のインストール手順を実行します。

単一の Ansible Playbook `install.yaml` で、次のタスクを実行します。

- NCP の準備
- OpenShift のインストール
- NCP のインストール

または、次の 2 つのセクションに記載された手順を使用して NCP と OpenShift をインストールすることもできます：[「CNI プラグイン、OVS、NCP Docker イメージのインストール」](#) および [「OpenShift Container Platform のインストール」](#)。

`install.yaml` Playbook の実行前に、`ncp_prep` および `ncp` Playbook ロールの必須パラメータとオプションパラメータを設定します。パラメータは [「CNI プラグイン、OVS、NCP Docker イメージのインストール」](#) に記載されています。

次のコマンドで Playbook を実行します。

```
ansible-playbook -i /PATH/TO/HOSTS/hosts install.yaml
```

CNI プラグイン、OVS、NCP Docker イメージのインストール

CNI (container network interface) プラグイン、OVS (Open vSwitch)、NCP Docker イメージは、OpenShift ノードにインストールする必要があります。このインストールは Ansible Playbook によって実行されます。

注： 単一の Playbook を使用して NCP と OpenShift をインストールする場合、この手順は必要ありません。[「単一の Playbook を使用した NCP と OpenShift のインストール」](#) を参照してください。

Playbook には、ノードに NSX-T リソースを設定する手順が含まれています。[章 2 「NSX-T リソースのセットアップ」](#) の説明に従って、NSX-T リソースを手動で設定することもできます。`perform_nsx_config` パラメータは、Playbook の実行時にリソースを設定するかどうかを指定します。

手順

- 1 `roles/ncp_prep/default/main.yaml` と `roles/nsx_config/default/main.yaml` のパラメータ値を更新します。CNI プラグイン RPM、OVS、対応するカーネル モジュールの RPM をダウンロードする URL も更新します。`uplink_port` は、ノード仮想マシンのアップリンク ポート vNIC の名前です。残りの変数は、次のような、NSX-T 管理プレーンの構成に関連する変数です。

指定する必要があるパラメータ：

- `perform_nsx_config`：リソース設定を行うかどうかを指定します。設定を手動で行う場合は `false` に設定します。この場合、`nsx_config` スクリプトは実行されません。
- `nsx_manager_ip`：NSX Manager の IP アドレス
- `nsx_edge_cluster_name`：Tier-0 ルーターで使用される Edge クラスタの名前
- `nsx_transport_zone_name`：オーバーレイ トランスポート ゾーンの名前

- **os_node_name_list** : ノード名のカンマ区切りリスト
node1、node2、node3 など
- **subnet_cidr** : 管理者がノード上の br-int に割り当てる IP の CIDR アドレス
- **vc_host** : vCenter Server の IP アドレス
- **vc_user** : vCenter Server 管理者のユーザー名
- **vc_password** : vCenter Server 管理者のパスワード
- **vms** : 仮想マシン名のカンマ区切りリスト。順番は **os_node_name_list** と一致させる必要があります。

次のパラメータには、デフォルト値が指定されています。これらは必要に応じて変更できます。

- **nsx_t0_router_name** : クラスタの Tier-0 論理ルーターの名前。デフォルト : **t0**
- **pod_ipblock_name** : ポッドの IP アドレス ブロックの名前。デフォルト : **podIPBlock**
- **pod_ipblock_cidr** : この IP アドレス ブロックの CIDR アドレス。デフォルト : **172.20.0.0/16**
- **snat_ippool_name** : SNAT の IP アドレス ブロックの名前。デフォルトは **externalIP** です。
- **snat_ippool_cidr** : この IP アドレス ブロックの CIDR アドレス。デフォルト : **172.30.0.0/16**
- **start_range** : この IP アドレス プールに対して指定された CIDR の開始 IP アドレス。デフォルト : **172.30.0.1**
- **end_range** : この IP アドレス プールに対して指定された CIDR の終了 IP アドレス。デフォルト : **172.30.255.254**
- **os_cluster_name** : OpenShift クラスタの名前。デフォルト : **occl-one**
- **nsx_node_ls_name** : ノードに接続されている論理スイッチの名前。デフォルト : **node_ls**
- **nsx_node_lr_name** : スイッチ **node_ls** の論理ルーターの名前。デフォルト : **node_lr**

nsx-config Playbook では、1 つの IP アドレス プールと 1 つの IP アドレス ブロックのみを作成できます。複数必要な場合は、手動で作成してください。

2 openshift-ansible-nsx ディレクトリに変更し、ncp_prep ロールを実行します。

```
ansible-playbook -i /PATH/T0/HOSTS/hosts ncp_prep.yaml
```

Playbook には、次の操作を実行する手順が含まれています。

- CNI プラグイン 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** です。<xxxxxxx> はビルド番号です。
- CNI プラグイン インストール 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 インストール ファイルをダウンロードしてインストールします。

ファイル名は `openvswitch-2.7.0.xxxxxxx-1.x86_64.rpm`、`openvswitch-kmod-2.7.0.xxxxxxx-1.el7.x86_64.rpm` です。<xxxxxxx> はビルド番号です。

- OVS が実行されていることを確認します。

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

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

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

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

- 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。
- NCP tar ファイルをダウンロードし、この tar ファイルから Docker イメージをロードします。
- `ncp-rbac.yaml` ファイルをダウンロードし、`apiVersion` を **v1** に変更します。
- NSX-T に論理トポロジおよび関連リソースを作成し、NCP が認識できるようにタグを作成します。
- NSX-T リソース情報を使用して `ncp.ini` を更新します。

次のステップ

OpenShift Container Platform をインストールします。「[「OpenShift Container Platform のインストール」](#)」を参照してください。

OpenShift Container Platform のインストール

OpenShift Container Platform (OCP) は、Docker と Kubernetes を統合する Platform as a Service (PaaS) 製品です。

注: 単一の Playbook を使用して NCP と OpenShift をインストールする場合、この手順は必要ありません。「[「単一の Playbook を使用した NCP と OpenShift のインストール」](#)」を参照してください。

OCP のインストールについては、<https://docs.openshift.com> を参照してください。

次のステップ

NCP と NSX Node Agent を実行します。「[「NCP と NSX Node Agent の実行」](#)」を参照してください。

NCP と NSX Node Agent の実行

NCP と NSX Node Agent を設定し、実行します。

手順

- 1 `roles/ncp/defaults/main.yaml` を編集して、OpenShift API サーバの IP アドレス、NSX Manager の IP アドレス、および NCP ReplicationController yaml と nsx-node-agent DaemonSet yaml をダウンロードする URL を指定します。
- 2 openshift-ansible-nsx ディレクトリから ncp ロールを実行します。

```
ansible-playbook -i /PATH/TO/HOSTS/hosts ncp.yaml
```

ncp ロールは次の処理を実行します。

- nsx-system プロジェクトが存在するかどうかを確認します。存在しない場合にはプロジェクトを作成します。

```
oc new-project nsx-system
```

- `ncp-rbac` yaml ファイルをダウンロードし、`apiVersion` を **v1** に変更します。
- NCP ポッドのサービス アカウントを作成し、NCP がアクセス可能なリソースを指定するクラスタ ロールを作成して、そのクラスタ ロールを NCP サービス アカウントにバインドします。
- nsx-node-agent ポッドのサービス アカウントを作成し、Node Agent がアクセス可能なリソースを指定するクラスタ ロールを作成して、そのクラスタ ロールを Node Agent サービス アカウントにバインドします。

```
oc apply -f /tmp/ncp-rbac.yaml
```

- 上記のサービス アカウントに関連付けられているトークンを取得し、`/etc/nsx-ujo/<service_account>_token` に保存します。

```
secret=`kubectl get serviceaccount ncp-svc-account -o yaml | grep -A1 secrets | tail
-n1 | awk {'print $3'}`
kubectl get secret $secret -o yaml | grep 'token:' | awk {'print $2'} | base64 -d
> /etc/nsx-ujo/ncp_token
secret=`kubectl get serviceaccount nsx-node-agent-svc-account -o yaml | grep -A1
secrets | tail -n1 | awk {'print $3'}`
kubectl get secret $secret -o yaml | grep 'token:' | awk {'print $2'} | base64 -d
> /etc/nsx-ujo/node_agent_token
```

- NCP の SecurityContextConstraint (SCC) yaml ファイルをダウンロードし、yaml に基づいて SCC を作成します。
- 作成した SCC を上記のサービス アカウントに追加します。

```
oadm policy add-scc-to-user ncp-scc -z ncp-svc-account
oadm policy add-scc-to-user ncp-scc -z nsx-node-agent-svc-account
```

- NCP ReplicationController (RC) と nsx-node-agent DaemonSet (DS) の yaml ファイルをダウンロードし、ConfigMap を更新します。
- NCP イメージをダウンロードしてロードします (nsx-node-agent も同じイメージを使用します)。

- サービス アカウントを設定し、NCP と nsx_node_agent に必要な SecurityContextConstraint を設定します。
- NCP ReplicationController と nsx-node-agent DaemonSet を作成します。

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。

セットアップに関する注意事項

OpenShift と NCP をセットアップする前に、次のことに注意してください。

- ポッドのラベル数は 11 個以下に、名前空間のラベル数は 12 個以下にする必要があります。
- OpenShift の内部用に追加したラベル、たとえば キーに openshift.io プリフィックスの付いているラベルなどは NCP で無視されます。そのため、関連する NSX リソースで作成される対応するタグは、ユーザーに表示されません。次は、OpenShift で使用されるラベル プリフィックスの一覧です。これらのプリフィックスで始まるラベル キーは使用しないでください。

```
openshift.io
pod-template
```

- ノードは、Kubelet health-checks などのポッドにアクセスする必要があります。ホストの管理インターフェイスがポッド ネットワークにアクセスできることを確認してください。
- 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、制限付きの anyuid SCC では、NET_ADMIN に権限は付与されません。NET_ADMIN を明示的に有効にする SCC や、ポッドを特権モードで実行する SCC には注意してください。信頼されていないコンテナでは、NET_RAW 機能を持たない anyuid SCC など、別の SCC を作成してください。この設定を行うには、SCC 定義で NET_RAW を「requiredDropCapabilities」リストに追加します。
- Ansible Playbook には、OpenShift 3.6 から 3.7 へのアップデートの際、インベントリ ファイルの **openshift_use_openshift_sdn=false** 設定を無視し、OpenShift のデフォルトの SDN をインストールするという不具合があります。この問題を回避するには、アップデート前に CNI プラグインを削除し、アップデート後に再インストールします。
- テストの場合にのみ、ポッド/コンテナに root アクセスを許可します。次のコマンドを実行するには、現在ログインしている OpenShift プロジェクトのすべてのポッドで root 権限が必要になります。

```
oc new-project test-project
oc project test-project
oadm policy add-scc-to-user anyuid -z default
```

- OpenShift レジストリを設定（追加）します。

```
oc login -u system:admin -n default
oadm registry --service-account=registry --config=/etc/origin/master/admin.kubeconfig
```

- OpenShift レジストリを削除します。

```
oc login -u system:admin -n default
oc delete svc/docker-registry dc/docker-registry
```

- Docker のデフォルト ブリッジ コンテナから、ノードの dnsmasq プロセスへの DNS 要求を許可する IPtables ファイアウォールルールはありません。これを手動で追加する必要があります。`/etc/sysconfig/iptables` を編集して、ファイルの最後にある COMMIT の前に次のルールを追加します。

```
-A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport 53 -j ACCEPT
-A OS_FIREWALL_ALLOW -p udp -m state --state NEW -m udp --dport 53 -j ACCEPT
COMMIT
```

- iptables、Docker、origin-node を再起動します (kube-proxy と kubelet を再起動します)。

```
systemctl restart iptables
systemctl restart docker
systemctl restart origin-node
```

- OpenShift の内部 Docker レジストリで、TLS 以外で OpenShift が機能するように許可する必要があります。通常、これは OpenShift Ansible インストーラによって自動的に追加されますが、機能していない可能性があります。`/etc/sysconfig/docker` を編集して、次を追加します。

```
INSECURE_REGISTRY='--insecure-registry 172.30.0.0/16'
```

- Docker を再起動します。

```
systemctl restart docker
```

- 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。
 - OpenShift 3.7、3.9 : ネットワーク ポリシーのルール句に、**namespaceSelector**、**podSelector**、および **ipBlock** のセクタが最大で 1 つ含まれている場合があります。
 - OpenShift 3.7 : 出力方向のルールおよび **ipBlock** CIDR セクタはサポートされません。これらは、OpenShift 3.9 でベータ版フィールドとして提供されています。

- Kubernetes の一部のバージョンで **subPath** に関連する問題があります (<https://github.com/kubernetes/kubernetes/issues/61076> を参照)。OpenShift バージョンにこの問題の修正が含まれていない場合は、**CreateContainerConfigError: failed to prepare subPath for volumeMount(CreateContainerConfigError: volumeMount のサブパスの準備に失敗しました)** というエラーが発生して、NCP ポッドの作成に失敗します。この問題を回避するには、NCP yaml から **subPath** の使用を削除します。特に、**subPath: ncp.ini** を含む行を削除して、**volumes** の設定を次の内容で置き換えます。

```
volumes:
- name: config-volume
  # ConfigMap nsx-ncp-config is expected to supply ncp.ini
  configMap:
    name: nsx-ncp-config
    items:
      - key: ncp.ini
        path: ncp.ini
```

この変更により、**/etc/nsx-uj0** ディレクトリ全体が読み取り専用になります。その結果、NCP は **etc/nsx-uj0** の下に一時ファイルを作成して証明書とプライベート キーの両方を 1 つのファイルに移動できなくなるため、証明書およびプライベート キーを使用して NSX-T と接続できなくなります。

ロード バランシング

NSX-T ロード バランサは、OpenShift と連携し、OpenShift Router として機能します。

NCP は OpenShift Route とエンドポイントのイベントを監視し、ルートの仕様に基づいて、ロード バランサ上のロード バランシングルールを設定します。結果的に、NSX-T ロード バランサはこのルールに基づき、適切なバックエンド ポッドにレイヤー 7 の受信トラフィックを転送します。

ロード バランシングの設定

ロード バランシングの設定には、Kubernetes LoadBalancer サービスや OpenShift Route の設定などが含まれます。また、NCP レプリケーション コントローラの設定も必要です。LoadBalancer サービスは、レイヤー 4 トラフィック向け、OpenShift Route は、レイヤー 7 トラフィック向けです。

Kubernetes LoadBalancer サービスを設定すると、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。

NSX-T のロード バランサを使用するには、ロード バランシングを NCP で設定する必要があります。**ncp_rc.yml** ファイルで次の操作を行います。

- 1 **use_native_loadbalancer** を **True** に設定します。
- 2 **pool_algorithm** を **WEIGHTED_ROUND_ROBIN** に設定します。
- 3 **lb_default_cert_path** と **lb_priv_key_path** が、それぞれ認証局 (CA) 署名証明書ファイルとプライベートキーファイルの完全パス名になるように設定します。CA 署名証明書を生成するサンプル スクリプトについては、以降を参照してください。また、NCP ポッドに、デフォルトの証明書とキーをマウントします。手順については、以降を参照してください。
- 4 (オプション) **service_size** に **SMALL**、**MEDIUM**、または **LARGE** を設定します。デフォルトは **SMALL** です。

レイヤー 7 ロード バランサの例

次の YAML ファイルでは、レイヤー 7 ロード バランシングを提供する 2 つのレプリケーション コントローラ (tea-rc および coffee-rc)、2 つのサービス (tea-svc および coffee-svc、2 つのルート (cafe-route-multi および cafe-route) を設定しています。

```
# RC
apiVersion: v1
kind: ReplicationController
metadata:
  name: tea-rc
spec:
  replicas: 2
  template:
    metadata:
      labels:
        app: tea
    spec:
      containers:
      - name: tea
        image: nginxdemos/hello
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: ReplicationController
metadata:
  name: coffee-rc
spec:
  replicas: 2
  template:
    metadata:
      labels:
        app: coffee
    spec:
      containers:
      - name: coffee
        image: nginxdemos/hello
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 80
---
# Services
apiVersion: v1
kind: Service
metadata:
  name: tea-svc
  labels:
    app: tea
spec:
  ports:
  - port: 80
```

```
    targetPort: 80
    protocol: TCP
    name: http
  selector:
    app: tea
---
apiVersion: v1
kind: Service
metadata:
  name: coffee-svc
  labels:
    app: coffee
spec:
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
      name: http
  selector:
    app: coffee
---
# Routes
apiVersion: v1
kind: Route
metadata:
  name: cafe-route-multi
spec:
  host: www.cafe.com
  path: /drinks
  to:
    kind: Service
    name: tea-svc
    weight: 1
  alternateBackends:
    - kind: Service
      name: coffee-svc
      weight: 2
---
apiVersion: v1
kind: Route
metadata:
  name: cafe-route
spec:
  host: www.cafe.com
  path: /tea-svc
  to:
    kind: Service
    name: tea-svc
    weight: 1
```

補注

- 本リリースでは、ロード バランシングは、OpenShift Route のリソースを介してのみサポートされ、タイプが **LoadBalancer** のサービス リソースではサポートされません。
- HTTPS トラフィックでは Edge ターミネーションのみがサポートされます。
- ワイルドカードを使用するサブドメインの指定がサポートされています。たとえば、**Subdomain** に **wildcardPolicy** が設定されていて、ホスト名が **wildcard.example.com** に設定されている場合、***. example.com** へのすべての要求に対してサービスが提供されます。
- 設定の誤りが原因で、Route イベントの処理中に NCP でエラーが発生する場合は、Route の YAML ファイルを修正し、Route リソースを削除して再作成する必要があります。
- NCP では、名前空間によるホスト名の所有は適用されません。
- Kubernetes クラスタあたり 1 つの LoadBalancer サービスがサポートされます。
- NSX-T 各 LoadBalancer 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。
- NSX-T ロード バランサには、異なるサイズを使用できます。NSX-T のロード バランサの構成については、『NSX-T 管理ガイド』を参照してください。

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 Route リソースによって定義されるポート合計 10 個。
- LoadBalancer サービスと Route 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 Route リソースによって定義されるポート合計 100 個。
- LoadBalancer サービスと Route 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 Route リソースによって定義されるポート合計 1000 個。
- LoadBalancer サービスと Route 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。

CA 署名証明書を作成するサンプル スクリプト

次のスクリプトによって、CA 署名証明書とプライベート キーが生成され、それぞれファイル <filename>.crt および <finename>.key として格納されます。**genrsa** コマンドで CA キーを生成します。CA キーは暗号化する必要があります。**aes256** などのコマンドを使用して、暗号化方式を指定できます。

```
#!/bin/bash
host="www.example.com"
filename=server

openssl genrsa -out ca.key 4096
openssl req -key ca.key -new -x509 -days 365 -sha256 -extensions v3_ca -out ca.crt -subj
"/C=US/ST=CA/L=Palo Alto/O=OS3/OU=Eng/CN=${host}"
openssl req -out ${filename}.csr -new -newkey rsa:2048 -nodes -keyout ${filename}.key -subj
"/C=US/ST=CA/L=Palo Alto/O=OS3/OU=Eng/CN=${host}"
openssl x509 -req -days 360 -in ${filename}.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out
${filename}.crt -sha256
```

デフォルトの証明書とキーの NCP ポッドへのマウント

証明書とプライベート キーは生成後、ホスト仮想マシンのディレクトリ **/etc/nsx-uj0** に配置されます。証明書とキー ファイルの名前がそれぞれ **lb-default.crt**、**lb-default.key** の場合、以下のように **ncp-rc.yaml** を編集して、ホスト上のこれらのファイルが、ポッドにマウントされるようにします。次はその例です。

```
spec:
  ...
  containers:
  - name: nsx-ncp
    ...
    volumeMounts:
    ...
    - name: lb-default-cert
      # Mount path must match nsx_v3 option "lb_default_cert_path"
      mountPath: /etc/nsx-uj0/lb-default.crt
    - name: lb-priv-key
      # Mount path must match nsx_v3 option "lb_priv_key_path"
      mountPath: /etc/nsx-uj0/lb-default.key
  volumes:
  ...
  - name: lb-default-cert
    hostPath:
```

```
    path: /etc/nsx-ujo/lb-default.crt
- name: lb-priv-key
  hostPath:
    path: /etc/nsx-ujo/lb-default.key
```

NSX-T Container Plug-in の管理

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 vmknics 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witch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 5-1. NCP コンテナの CLI コマンド

タイプ	コマンド
状態	get ncp-master status
状態	get ncp-nsx status
状態	get ncp-watcher <watcher-name>
状態	get ncp-watchers
状態	get ncp-k8s-api-server status
状態	check projects
状態	check project <project-name>
キャッシュ	get project-cache <project-name>
キャッシュ	get project-caches

表 5-1. NCP コンテナの CLI コマンド (続き)

タイプ	コマンド
キャッシュ	get namespace-cache <namespace-name>
キャッシュ	get namespace-caches
キャッシュ	get pod-cache <pod-name>
キャッシュ	get pod-caches
キャッシュ	get ingress-caches
キャッシュ	get ingress-cache <ingress-name>
キャッシュ	get ingress-controllers
キャッシュ	get ingress-controller <ingress-controller-name>
キャッシュ	get network-policy-caches
キャッシュ	get network-policy-cache <pod-name>
サポート	get ncp-log file <filename>
サポート	get ncp-log-level
サポート	set ncp-log-level <log-level>
サポート	get support-bundle file <filename>
サポート	get node-agent-log file <filename>
サポート	get node-agent-log file <filename> <node-name>

表 5-2. 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 5-3. 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 :

```
kubecall> 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 :

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

例 1 :

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

例 2 :

```
kubecall> get ncp-watchers
pod:
Average event processing time: 1145 msec (in past 3600-sec window)
Current watcher started time: Mar 02 2017 10:51:37 PST
Number of events processed: 1 (in past 3600-sec window)
Total events processed by current watcher: 1
Total events processed since watcher thread created: 1
Total watcher recycle count: 0
Watcher thread created time: Mar 02 2017 10:51:37 PST
Watcher thread status: Up

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

```
Total watcher recycle count: 0
Watcher thread created time: Mar 02 2017 10:51:37 PST
Watcher thread status: Up
```

ingress:

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

service:

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

- NCP 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 :

```
kubecall> 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 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 :

```
kubecall> 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

kubecall> 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

kubecall> 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-uw2lv:
    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

kubensode> get pod-cache nsx.default.nginx-rc-ug2lv
  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

```

- 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 :

```

kubensode> 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
```

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 1 :

```
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
```

例 2 :

```
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
```

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 :

```
kubecall> 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 1 :

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

例 2 :

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

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

■ ノードの OVS フローのダンプ

```
dump ovs-flows
```

例 :

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

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
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

エラー コード	説明
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
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

エラー コード	説明
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

エラー コード	説明
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
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

エラー コード	説明
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
NCP05011	Kubernetes エンドポイントの処理に失敗しました

エラー コード	説明
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

OpenShift のエラー コード

エラー コード	説明
NCP07001	OC ルートの処理に失敗しました
NCP07002	OC ルート ステータスの更新に失敗しました