

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 4 月 10 日

VMware NSX-T 2.1.0.1、2.1.2

VMware NSX-T Data Center 2.1



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

互換性要件 5

インストールの概要 6

2 NSX-T リソースのセットアップ 7

NSX-T リソースの設定 7

Tier-0 論理ルーターの作成と設定 10

3 NSX-T Container Plug-in と OpenShift のセットアップ 12

OpenShift 仮想マシンの展開 12

Ansible の hosts ファイルの準備 12

CNI プラグインと OVS 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 17

セットアップに関する注意事項 18

4 NSX-T Container Plug-in の管理 21

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 21

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 22

CIF 接続論理ポート 22

CLI コマンド 23

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。

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。

- [互換性要件](#)
- [インストールの概要](#)

互換性要件

NSX-T Container Plug-in には、次の互換性要件があります。

ソフトウェア製品	バージョン
コンテナ ホストの仮想マシンのハイパーバイザー	<ul style="list-style-type: none">■ ESXi 6.5■ ESXi 6.5 Update 1■ RHEL KVM 7.3、7.4
コンテナ ホストのオペレーティング システム	RHEL 7.3、7.4
Platform as a Service	OpenShift 3.7 (NCP 2.1、2.1.0.1) OpenShift 3.7、3.9 (NCP 2.1.2)
ゲスト vSwitch	OVS 2.6、2.7、2.8

インストールの概要

以下の手順に従って、NCP のインストールと構成を行います。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 ブロックまたは IP アドレス プールを作成します。
- 7 OpenShift 仮想マシンをデプロイします。
- 8 Ansible hosts ファイルを準備します。
- 9 各ノードに CNI プラグインと OVS (Open vSwitch) をインストールします。
- 10 OpenShift Container Platform をインストールします。
- 11 NCP と NSX Node Agent を実行します。

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、[「CNF プラゲインと OVS 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。

NCP 2.1 および 2.1.0.1 で NSX-T リソースを設定するには、タグを使用する必要があります。NCP 2.1.2 以降では、設定ファイル `ncp.ini` の UUID または名前を使用して NSX-T リソースを設定する必要があります。

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 (Fabric)] - [トランスポート ゾーン (Transport Zones)] 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 2.1 および 2.1.0.1) トランスポート ゾーンに 1 つ以上の `{'ncp/cluster': '<cluster_name>'}` タグを付けて 1 つ以上のクラスタに固有にするか、`{'ncp/shared_resource': 'true'}` をタグ付けしてクラスタ間で共有します。タグ `ncp/cluster` が存在する場合、タグ内で名前が指定されているクラスタは

`ncp/cluster` タグ付きリソースのみを使用し、`ncp/shared_resource` タグ付きリソースは使用しません。クラスタ名は、`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。

(NCP 2.1.2) `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 (Routing)] - [ルーター (Routers)] 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 2.1 および 2.1.0.1) ルーターに 1 つ以上の `{'ncp/cluster': '<cluster_name>'}` タグを付けて 1 つ以上のクラスタに固有にするか、`{'ncp/shared_resource': 'true'}` をタグ付けしてクラスタ間で共有します。タグ `ncp/cluster` が存在する場合、タグ内で名前が指定されているクラスタは `ncp/cluster` タグ付きリソースのみを使用し、`ncp/shared_resource` タグ付きリソースは使用しません。クラスタ名は、`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。

(NCP 2.1.2) `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

データ トラフィック用にノードが使用する 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 (Switching)] - [スイッチ (Switches)] 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、それらにノード vNIC を接続します。論理ポートには、`{'ncp/cluster': '<cluster_name>'}` および `{'ncp/node_name': '<node_name>'}` のタグを付ける必要があります。`<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 (IPAM)] 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 2.1 および 2.1.0.1) IP アドレス ブロックに 1 つ以上の `{'ncp/cluster': '<cluster_name>'}` タグを付けて 1 つ以上のクラスタに固有にするか、`{'ncp/shared_resource': 'true'}` をタグ付けしてクラスタ間で共有します。タグ `ncp/cluster` が存在する場合、タグ内で名前が指定されているクラスタは `ncp/cluster` タグ付きリソースのみを使用し、`ncp/shared_resource` タグ付きリソースは使用しません。クラスタ名は、`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。

(NCP 2.1.2) `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 2.1 および 2.1.0.1) IP アドレス ブロックに `ncp/cluster` タグだけでなく、`{'ncp/no_snat': 'true'}` をタグ付けします。以前のリリースからのアップグレードで `{'ncp/no_snat': '<cluster_name>'}` タグがある場合は、このタグを手動で `{'ncp/no_snat': 'true'}` に変更する必要があります。

(NCP 2.1.2) `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 ブロックを作成する場合には、NCP を再起動する必要があります。再起動しない場合、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

IP アドレス プールは、SNAT ルールを通じてポッドの IP アドレスを変換する際、または SNAT/DNAT ルールを通じて Ingress Controller を公開する際の IP アドレスの割り当てで、OpenStack のフローティング IP アドレスと同じように使用されます。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 (Inventory)] - [グループ (Groups)] - [IP アドレス プール (IP POOL)] 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 2.1 および 2.1.0.1) プールに `{'ncp/external': 'true'}` をタグ付けします。さらに、プールに 1 つ以上の `{'ncp/cluster': '<cluster_name>'}` タグを付けて 1 つ以上のクラスタに固有にするか、`{'ncp/shared_resource': 'true'}` をタグ付けしてクラスタ間で共有します。タグ `ncp/cluster` が存在する場合、タグ内で名前が指定されているクラスタは `ncp/cluster` タグ付きリソースのみを使用し、`ncp/shared_resource` タグ付きリソースは使用しません。クラスタ名は、`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。

(NCP 2.1.2) `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。

サービスに注釈を追加して、特定のサービスの SNAT を設定することもできます。次に例を示します。

```
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。このルールのソース 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 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 ルールの方がプロジェクトの優先順位よりも大きくなります。

- 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。

(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 (Firewall)] - [全般 (General)] 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 2.1 および 2.1.0.1) `{'ncp/fw_sect_marker': 'bottom'}` および `{'ncp/fw_sect_marker': 'top'}` にそれぞれ異なるセクションをタグ付けします。

(NCP 2.1.2) `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 が隔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](#)
- [CNI プラグインと OVS 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 つ以上のマスター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 クラスター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。
- 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。次に例を示します。

```
[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 アドレスを識別します。使用するインターフェイスがデフォルトでは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。

- 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

```

- [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（<https://docs.openshift.com> で「高度なインストール」を検索）を参照してください。次に例を示します。

```

# 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 プラグインと OVS のインストール」](#) を参照してください。

CNI プラグインと OVS のインストール

コンテナ ネットワーク インターフェイス (CNI) プラグインと Open vSwitch (OVS) 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。

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。

- `perform_nsx_config` : リソース構成を実行するかどうかを示します。構成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_config_script_path` : `nsx_config.py` スクリプトの絶対パス
- `nsx_cert_file_path` : NSX-T クライアント証明書ファイルの絶対パス
- `nsx_manager_ip` : NSX Manager の IP アドレス
- `nsx_edge_cluster_name` : Tier-0 ルーターで使用する 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
- `nsx_t0_router_name` : クラスタの Tier-0 論理ルーターの名前
- `pod_ipblock_name` : ポッドの IP アドレス ブロックの名前
- `pod_ipblock_cidr` : この IP アドレス ブロックの CIDR アドレス
- `snat_ippool_name` : SNAT の IP アドレス ブロックの名前
- `snat_ippool_cidr` : この IP アドレス ブロックの CIDR アドレス
- `start_range` : この IP プールに対して指定された CIDR の開始 IP アドレス
- `end_range` : この IP プールに対して指定された CIDR の終了 IP アドレス
- `os_cluster_name` : OpenShift クラスタの名前
- `os_node_name_list` : ノード名のカンマ区切りリスト
node1、node2、node3 など
- `nsx_node_ls_name` : ノードに接続されている論理スイッチの名前
- `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。

この名前は、作成した NSX-T リソースと一致させる必要があります。一致していない場合、指定した名前でのリソースが作成されます。Playbook は冪等性があるため、何度実行しても結果が変わることはありません。指定された名前のリソースを確認して、適切にタグ付けを行います。

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、Docker イメージをこの tar ファイルからロードします。

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。

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ー ロールを作成して、そのクラスター ロールを NCP サービス アカウントとバインドします。
- 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ー ロールを作成して、そのクラスター ロールを Node Agent サービス アカウントとバインドします。

```
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 yml 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 サーバにセッション維持が可能な 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。
- サービスのロード バランシングを設定する場合は、OpenShift ルーターを使用する必要があります。NSX-T ロード バランサはサポートされていません。
- 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 機能のない 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、CNI プラグインをアップデート前に削除し、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 oc 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。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
```

- OpenShift ルーター (HA-Proxy N/S LB) を設定（追加）します。

```
oc login -u system:admin -n default
oadm router router --replicas=2 --service-account=router
```

- 作成されたルーターを削除します。

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

- サンプルの Ruby ベースの 2 層アプリケーションを作成します。

```
oc login -u system:admin -n default
oc oc new-project nsx
oc process -n openshift mysql-ephemeral -v DATABASE_SERVICE_NAME=database | oc create
-f -
oc new-app centos/ruby-22-centos7~https://github.com/openshift/ruby-hello-world.git
oc expose service ruby-hello-world
oc env dc database --list | oc env dc ruby-hello-world -e -
```

- ネットワーク ポリシーに対する NCP のサポートは Kubernetes によって提供されるサポートと同じであり、OpenShift で使用される Kubernetes バージョンに応じて異なります。
 - 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** のセレクトが最大で 1 つ含まれている可能性があります。
 - OpenShift 3.7 - Egress (出力方向) ルールおよび **ipBlockCIDR** セレクトのサポートは利用できません。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
```

この変更の影響により、**/etc/nsx-ujio** 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-ujio** の下に一時ファイルを作成して証明書とプライベート キーの両方を単一のファイルに移動することができなくなるため、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。

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](#)

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 4-1. NCP コンテナの CLI コマンド

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

表 4-1. 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 4-2. NSX Node Agent コンテナの CLI コマンド

タイプ	コマンド
ステータス	get node-agent-hyperbus status
キャッシュ	(NCP 2.1、2.1.0.1) get app-cache <app-name> (NCP 2.1.2) get container-cache <container-name>
キャッシュ	(NCP 2.1、2.1.0.1) get app-caches (NCP 2.1.2) get container-caches

表 4-3. NSX Kube-Proxy コンテナの CLI コマンド

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

NCP コンテナのステータス コマンド

- NCP マスターのステータスを表示します。

```
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 の間の接続ステータスを表示します。

```
get ncp-nsx status
```

例 :

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

- Ingress、名前空間、ポッド、サービスの監視ステータスを表示します。

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

例 1 :

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

例 2 :

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

```
Watcher thread status: Up
```

```
namespace:
```

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

```
ingress:
```

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

```
service:
```

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

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

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

例 :

```
kubnode> 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 :

```
kubnode> 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

kubenode> 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 :

```
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

kubecfg> 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

kubecfg> 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 :

```

kubecfg> get pod-caches
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

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-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 :

```

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

```

```

kubenset> 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 :

```

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

```

- NCP ログをファイルストアに保存します。

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

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

例 :

```

kubecall>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 コンテナのステータス コマンド

- このノードの Node Agent と HyperBus 間の接続ステータスを表示します。

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

例 :

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

NSX 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。

```
(NCP 2.1, 2.1.0.1) get app-cache <app-name>
(NCP 2.1, 2.1.0.1) get app-caches
(NCP 2.1.2) get container-cache <container-name>
(NCP 2.1.2) 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 :

```
kubecall> 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 コンテナのステータス コマンド

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

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

例 :

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

- Kube プロキシの監視ステータスを表示します。

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

例 1 :

```
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

```