适用于 Kubernetes 和 Cloud Foundry 的 NSX Container Plug-in - 安装和管理指南

VMware NSX Container Plug-in 2.4、2.4.1

VMware NSX-T Data Center 2.4

vmware®

您可以从 VMware 网站下载最新的技术文档：

https://docs.vmware.com/cn/。

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

威睿信息技术（中国）有限公司
北京办公室
北京市
朝阳区新源南路 8 号
启皓北京东塔 8 层 801
www.vmware.com/cn

上海办公室
上海市
淮海中路 333 号
瑞安大厦 804-809 室
www.vmware.com/cn

广州办公室
广州市
天河路 385 号
太古汇一座 3502 室
www.vmware.com/cn

目录

适用于 Kubernetes 和 Cloud Foundry 的 NSX Container Plug-in - 安装和管理指南

本指南介绍了如何安装和管理 NSX Container Plug-in (NCP) 以提供 NSX-T Data Center 与 Kubernetes 之间以及 NSX-T Data Center 与 Pivotal Cloud Foundry (PCF) 之间的集成。

目标读者

本指南适用于系统和网络管理员。假定用户熟悉 NSX-T Data Center、Kubernetes 和 Pivotal Cloud Foundry 的安装和管理。

VMware 技术出版物术语表

VMware 技术出版物提供了一个术语表，其中包含一些您可能不熟悉的术语。有关 VMware 技术文档中使用的术语的定义，请访问 http://www.vmware.com/support/pubs。

NSX Container Plug-in 概述

1

NSX Container Plug-in(NCP) 提供 NSX-T Data Center 与容器协调器（如 Kubernetes）之间的集成以及 NSX-T Data Center 与基于容器的 PaaS（平台即服务）产品（如 OpenShift 和 Pivotal Cloud Foundry）之间的集成。本指南介绍了如何设置 NCP 以与 Kubernetes 和 Pivotal Cloud Foundry 集成。

NCP 的主要组件在容器中运行，并与 NSX Manager 和 Kubernetes 控制层面进行通信。NCP 调用 NSX API 以监控对容器和其他资源的更改以及管理网络资源，如容器的逻辑端口、交换机、路由器和安全组。

NSX CNI 插件在每个 Kubernetes 节点上运行。它监控容器生命周期事件，将容器接口连接到客户机 vSwitch，并对客户机 vSwitch 进行编程以标记和转发容器接口和 vNIC 之间的容器流量。

NCP 提供了以下功能：

- 自动为 Kubernetes 群集创建 NSX-T Data Center 逻辑拓扑，并为每个 Kubernetes 命名空间创建一个单独的逻辑网络。
- 将 Kubernetes pod 连接到逻辑网络，并分配 IP 和 MAC 地址。
- 支持网络地址转换 (NAT) 并为每个 Kubernetes 命名空间分配一个单独的 SNAT IP。

  注 配置 NAT 时，转换后 IP 的总数不能超过 1000。

- 使用 NSX-T Data Center 分布式防火墙实现 Kubernetes 网络策略。
  - 支持输入和输出网络策略。
  - 支持网络策略中的 `IPBlock` 选择器。
  - 为网络策略指定标签选择器时，支持 `matchLabels` 和 `matchExpression`。
  - 支持在其他命名空间中选择 pod。
- 实现 `ClusterIP` 类型的 Kubernetes 服务和 `LoadBalancer` 类型的服务。
- 使用 NSX-T 第 7 层负载平衡器实现 Kubernetes Ingress。
  - 通过 TLS Edge 终止支持 HTTP Ingress 和 HTTPS Ingress。
  - 支持 Ingress 默认后端配置。
  - 支持 Ingress URI 重写。

- 在 NSX-T Data Center 逻辑交换机端口上为命名空间、pod 名称和 pod 标签创建标记，并允许管理员根据标记定义 NSX-T 安全组和策略。

在该版本中，NCP 支持单个 Kubernetes 群集。您可以具有使用相同 NSX-T Data Center 部署的多个 Kubernetes 群集，每个群集均有不同的 NCP 实例。

本章讨论了以下主题：

兼容性要求

NSX Container Plug-in (NCP) 对 Kubernetes 环境和 Pivotal Cloud Foundry (PCF) 环境具有以下兼容性要求。

表 1-1. Kubernetes 环境的兼容性要求

软件产品	版本
NSX-T Data Center	2.3、2.4
容器主机虚拟机的管理程序	■ 支持的 vSphere 版本 ■ RHEL KVM 7.4、7.5、7.6 ■ Ubuntu KVM 16.04
容器主机操作系统	■ RHEL 7.5、7.6 ■ Ubuntu 16.04 ■ CentOS 7.4、7.5
容器协调器	Kubernetes 1.12、1.13
容器主机 Open vSwitch	2.9.1（随 NSX-T Data Center 2.3.x 打包）、2.10.2（随 NSX-T Data Center 2.4.0 打包）

表 1-2. Cloud Foundry 环境的兼容性要求

软件产品	版本
容器主机虚拟机的管理程序	■ 支持的 vSphere 版本
容器协调器	■ Pivotal Application Service 2.3.x 和 Pivotal Operations Manager 2.3.x ■ Pivotal Application Service 2.4.x（2.4.0 除外）和 Pivotal Operations Manager 2.4.x（2.4.0 除外）

安装概述

在已安装 Kubernetes 的环境中，安装和配置 NCP 通常包括以下步骤。要成功执行这些步骤，您必须熟悉 NSX-T Data Center 和 Kubernetes 安装和管理。

1 安装 NSX-T Data Center。

2 创建一个覆盖网络传输区域。

3 创建一个覆盖网络逻辑交换机并将 Kubernetes 节点连接到该交换机。

4 创建一个 Tier-0 逻辑路由器。

5 为 Kubernetes pod 创建 IP 块。

6 为 SNAT（源网络地址转换）创建 IP 池。

7 在每个节点上安装 NSX CNI（容器网络接口）插件。

8 在每个节点上安装 OVS (Open vSwitch)。

9 为 Kubernetes 节点配置 NSX-T 网络。

10 将 NSX 节点代理安装为 DaemonSet。

11 将 NCP 安装为 ReplicationController。

12 在 NCP pod 中挂载安全证书。

在 Kubernetes 环境中升级 NCP

本节介绍如何在 Kubernetes 环境中将 NCP 升级到 2.4.0。

步骤

1 使用以下命令升级 NCP ReplicationController（将 `<image>` 替换为映像的实际名称）。

```
kubectl rolling-update nsx-ncp -n nsx-system --image=<image>
```

2 使用以下命令升级 NSX 节点代理 daemonSet（将 `<image>` 替换为映像的实际名称）。

```
kubectl set image ds nsx-node-agent -n nsx-system nsx-node-agent=<image>
kubectl set image ds nsx-node-agent -n nsx-system nsx-kube-proxy=<images>
kubectl rollout status ds/nsx-node-agent -nnsx-system
```

3 使用以下命令将 CNI DEB/RPM 软件包升级到 2.4.0（将 `<cni deb>` 和 `<cni rpm>` 替换为软件包的实际名称）。

在 Ubuntu 上：

```
dkpg -i <cni deb>
```

在 RHEL 或 CentOS 上：

```
rpm -U <cni rpm>
```

4 （可选） 将 NSX-T Data Center 升级到 2.4。

如果管理程序是 ESXi，则在升级 NSX-T Data Center 之前，将 ESXi 从 6.5 至少升级到 6.5p03 或者从 6.7 至少升级到 6.7ep6。

5 （可选） 升级管理程序（KVM 或裸机容器）。

6 （可选） 升级容器主机（RHEL、Ubuntu 或 CentOS）。

7 （可选） 升级 Kubernetes。

8 （可选） 升级 OVS。

对于裸机容器，升级 NSX-T Data Center 也会升级 OVS，因此无需执行此步骤。

执行此步骤时，nsx-kube-proxy 和 nsx-node-agent 之间可能会出现暂时性通信故障。这是预期行为，并不表示存在问题。

在 Pivotal Cloud Foundry 环境中升级 NCP

本节介绍如何在 Pivotal Cloud Foundry 环境中将 NCP 升级到 2.4.0。

步骤

1 将 NCP 或 NSX-T tile 包升级到 2.4.0。

2 （可选） 升级 Pivotal Operations Manager，然后升级 Pivotal Application Service。

3 （可选） 将 NSX-T Data Center 升级到 2.4。

如果管理程序是 ESXi，则在升级 NSX-T Data Center 之前，将 ESXi 从 6.5 至少升级到 6.5p03 或者从 6.7 至少升级到 6.7ep6。

4 （可选） 升级管理程序（KVM 或裸机容器）。

设置 NSX-T 资源

2

在安装 NSX Container Plug-in 之前，您需要设置某些 NSX-T Data Center 资源。

本章讨论了以下主题：

配置 NSX-T 资源

您需要配置的 NSX-T Data Center 资源包括覆盖网络传输区域、Tier-0 逻辑路由器、用于连接节点虚拟机的逻辑交换机、Kubernetes 节点的 IP 块以及 SNAT 的 IP 池。

重要事项 如果与 NSX-T Data Center 2.4 或更高版本一起运行，则必须使用**高级网络和安全**选项卡配置 NSX-T 资源。

在 NCP 配置文件 `ncp.ini` 中，使用 UUID 或名称指定 NSX-T Data Center 资源。

覆盖网络传输区域

登录到 NSX Manager，然后导航到**系统 > Fabric > 传输区域**。查找用于容器网络的覆盖网络传输区域，或者创建新的传输区域。

通过在 `ncp.ini` 的 `[nsx_v3]` 部分中设置 `overlay_tz` 选项来指定集群的覆盖网络传输区域。此步骤是可选的。如果未设置 `overlay_tz`，NCP 将自动从 Tier-0 路由器检索覆盖网络传输区域 ID。

Tier-0 逻辑路由

登录到 NSX Manager，然后导航到**高级网络和安全 > 网络 > 路由器**。查找用于容器网络的路由器，或者创建新的路由器。

通过在 `ncp.ini` 的 `[nsx_v3]` 部分中设置 `tier0_router` 选项来指定集群的 Tier-0 逻辑路由器。

注 必须在活动-备用模式下创建路由器。

逻辑交换机

节点用于数据流量的 vNIC 必须连接到覆盖网络逻辑交换机。不要求将节点的管理接口连接到 NSX-T Data Center，但这样做将简化设置过程。通过登录到 NSX Manager，然后导航到**高级网络和安全 > 网络 > 交换 > 交换机**，可以创建逻辑交换机在交换机上创建逻辑端口，并将节点 vNIC 连接到这些端口。逻辑端口必须具有以下标记：

- 标记：`<cluster_name>`，范围：`ncp/cluster`
- 标记：`<node_name>`，范围：`ncp/node_name`

`<cluster_name>` 值必须与 `ncp.ini` 的 `[coe]` 部分中的 `cluster` 选项值匹配。

Kubernetes pod 的 IP 块

登录到 NSX Manager，然后导航到**高级网络和安全 > 网络 > IPAM** 以创建一个或多个 IP 块。使用 CIDR 格式指定 IP 块。

通过在 `ncp.ini` 的 `[nsx_v3]` 部分中设置 `container_ip_blocks` 选项来指定 Kubernetes pod 的 IP 块。

您也可以专门为非 SNAT 命名空间（对于 Kubernetes）或集群（对于 PCF）创建 IP 块。

通过在 `ncp.ini` 的 `[nsx_v3]` 部分中设置 `no_snat_ip_blocks` 选项来指定非 SNAT IP 块。

如果在 NCP 运行时创建非 SNAT IP 块，您必须重新启动 NCP。否则，NCP 将继续使用共享 IP 块，直到这些块用尽。

注 在创建 IP 块时，前缀不能大于 NCP 配置文件 `ncp.ini` 中的 `subnet_prefix` 参数值。有关详细信息，请参见 ncp-rc.yml 中的 ncp.ini 的 ConfigMap。

SNAT 的 IP 池

将使用 NSX Manager 中的 IP 池分配 IP 地址，这些地址将用于通过 SNAT 规则转换 pod IP 以及通过 SNAT/DNAT 规则公开 Ingress 控制器，就像 Openstack 浮动 IP 一样。这些 IP 地址也称为外部 IP。

多个 Kubernetes 集群使用相同的外部 IP 池。每个 NCP 实例将该池的一部分用于它管理的 Kubernetes 集群。默认情况下，将使用 pod 子网的相同子网前缀。要使用不同的子网大小，请更新 `ncp.ini` 的 `[nsx_v3]` 部分中的 `external_subnet_prefix` 选项。

可以通过在 `ncp.ini` 的 `[nsx_v3]` 部分中设置 `external_ip_pools` 选项来指定 SNAT 的 IP 池。

可以通过更改配置文件并重新启动 NCP 来更改为不同的 IP 池。

将 SNAT IP 池限制到特定的 Kebernetes 命名空间或 PCF 组织

可以通过在 SNAT IP 池中添加以下标记指定可为哪些 Kubernetes 命名空间或 PCF 组织分配 IP 池中的 IP。

- 对于 Kubernetes 命名空间：`scope: ncp/owner, tag: ns:<namespace_UUID>`
- 对于 PCF 组织：`scope: ncp/owner, tag: org:<org_UUID>`

可以使用以下命令之一获取命名空间或组织 UUID：

```
kubectl get ns –o yaml
cf org <org_name> ––guid
```

请注意以下事项：

- 每个标记应指定一个 UUID。可以为同一个池创建多个标记。
- 如果根据旧标记为某些命名空间或组织分配 IP 后更改了标记，则 Kubernetes 服务或 PCF 应用程序的 SNAT 配置更改或 NCP 重新启动之前，将不会回收这些 IP。
- 命名空间和 PCF 组织所有者标记是可选的。如果没有这些标记，任何命名空间或 PCF 组织的 IP 都可以从 SNAT IP 池分配。

为服务配置 SNAT IP 池

可以通过向服务添加注释来为此服务配置 SNAT IP 池。例如，

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

`ncp/snat_pool` 指定的 IP 池必须具有标记 `scope: ncp/owner, tag: cluster:<cluster_name>`。

NCP 将为此服务配置 SNAT 规则。规则的源 IP 为后端 pod 集。目标 IP 是从指定外部 IP 池分配的 SNAT IP。如果 NCP 配置 SNAT 规则时出错，该服务中将添加 `ncp/error.snat: <error>` 注释。可能的错误包括：

- IP_POOL_NOT_FOUND - 在 NSX Manager 中找不到 SNAT IP 池。
- IP_POOL_EXHAUSTED - SNAT IP 池已用尽。
- IP_POOL_NOT_UNIQUE - `ncp/snat_pool` 指定的池引用 NSX Manager 的多个池。
- SNAT_POOL_ACCESS_DENY - 池的所有者标记与发送分配请求的服务的命名空间不匹配。
- SNAT_RULE_OVERLAPPED - 创建新的 SNAT 规则，但 SNAT 服务的 pod 同时也属于其他 SNAT 服务，即，同一个 pod 有多个 SNAT 规则。
- POOL_ACCESS_DENIED - `ncp/snat_pool` 指定的 IP 池没有标记 `scope: ncp/owner, tag: cluster:<cluster_name>`，或池的所有者标记与发送分配请求的服务的命名空间不匹配。

请注意以下事项：

- 配置服务之前，`ncp/snat_pool` 指定的池应该已存在于 NSX-T Data Center 中。
- 在 NSX-T Data Center 中，服务 SNAT 规则的优先级高于项目 SNAT 规则。

- 如果 pod 配置了多个 SNAT 规则，则只有一个起作用。
- 可以通过更改注释并重新启动 NCP 来更改为不同的 IP 池。

为命名空间配置 SNAT IP 池

可以通过向命名空间添加注释来为此命名空间配置 SNAT IP 池。例如，

```
apiVersion: v1
kind: Namespace
metadata:
  name: ns-sample
  annotations:
    ncp/snat_pool: <external IP pool ID or name>
...
```

NCP 将为此命名空间配置 SNAT 规则。规则的源 IP 为后端 pod 集。目标 IP 是从指定外部 IP 池分配的 SNAT IP。如果 NCP 配置 SNAT 规则时出错，该命名空间中将添加“`ncp/error.snat: <error>`”注释。可能的错误包括：

- IP_POOL_NOT_FOUND - 在 NSX Manager 中找不到 SNAT IP 池。
- IP_POOL_EXHAUSTED - SNAT IP 池已用尽。
- IP_POOL_NOT_UNIQUE - `ncp/snat_pool` 指定的池引用 NSX Manager 的多个池。
- POOL_ACCESS_DENIED - `ncp/snat_pool` 指定的 IP 池没有标记 `scope: ncp/owner, tag: cluster:<cluster_name>`，或池的所有者标记与发送分配请求的命名空间不匹配。

请注意以下事项：

- 只能在注释中指定一个 SNAT IP 池。
- 无需在 `ncp.ini` 中配置 SNAT IP 池。
- `ncp/snat_pool` 指定的 IP 池必须具有标记 `scope: ncp/owner, tag: cluster:<cluster_name>`。
- `ncp/snat_pool` 指定的 IP 池也可以具有命名空间标记 `scope: ncp/owner, tag: ns:<namespace_UUID>`。
- 如果缺少 `ncp/snat_pool` 注释，则命名空间会将 SNAT IP 池用于集群。
- 可以通过更改注释并重新启动 NCP 来更改为不同的 IP 池。

为 PAS 应用程序配置 SNAT 池

默认情况下，NCP 为 PAS (Pivotal Application Service) 组织配置 SNAT IP。可以通过为应用程序创建包含 SNAT IP 池信息的 `manifest.xml` 来为应用程序配置 SNAT IP。例如，

```
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 将为此应用程序配置 SNAT 规则。规则的源 IP 是实例的 IP 集，其目标 IP 是从外部 IP 池分配的 SNAT IP。请注意以下事项：

- 推送应用程序之前，NCP_SNAT_POOL 指定的池应该已存在于 NSX-T Data Center 中。
- 应用程序 SNAT 规则的优先级高于组织 SNAT 规则。
- 只能为应用程序配置一个 SNAT IP。
- 可以通过更改配置并重新启动 NCP 来更改为不同的 IP 池。

为 PCF 版本 3 配置 SNAT

在 PCF 版本 3 中，可以通过以下两种方法之一配置 SNAT：

- 创建应用程序时，在 `manifest.yml` 中配置 NCP_SNAT_POOL。

  例如，应用程序称为 `bread`，`manifest.yml` 具有以下信息：

  ```
  applications:
  - name: bread
    stack: cflinuxfs2
    random-route: true
    env:
      NCP_SNAT_POOL: AppSnatExternalIppool
    processes:
    - type: web
      disk_quota: 1024M
      instances: 2
      memory: 512M
      health-check-type: port
    - type: worker
      disk_quota: 1024M
      health-check-type: process
      instances: 2
      memory: 256M
      timeout: 15
  ```

  运行以下命令：

  ```
  cf v3-push bread
  cf v3-apply-manifest -f manifest.yml
  cf v3-apps
  cf v3-restart bread
  ```

- 使用 `cf v3-set-env` 命令配置 NCP_SNAT_POOL。

运行以下命令（假定应用程序称为 app3）：

```
cf v3-set-env app3 NCP_SNAT_POOL AppSnatExternalIppool
(optional) cf v3-stage app3 -package-guid <package-guid> (You can get package-guid with "cf v3-packages app3".)
cf v3-restart app3
```

（可选）（仅适用于 Kubernetes）防火墙标记区域

要允许管理员创建防火墙规则且不干扰 NCP 基于网络策略创建的防火墙区域，请登录到 NSX Manager，导航到**安全 > 分布式防火墙 > 常规**并创建两个防火墙区域。

通过在 ncp.ini 的 [nsx_v3] 部分中设置 bottom_firewall_section_marker 和 top_firewall_section_marker 选项来指定标记防火墙区域。

底部防火墙区域必须在顶部防火墙区域的下方。创建这些防火墙区域后，NCP 创建的用于隔离的所有防火墙区域将创建在底部防火墙区域的上方，NCP 创建的用于策略的所有防火墙区域将创建在顶部防火墙区域的下方。如果未创建这些标记区域，则将在底部创建所有隔离规则，在顶部创建所有策略区域。不支持每个集群中多个标记防火墙区域采用相同的值，这将会导致错误。

在 Kubernetes 环境中安装 NCP

3

安装 NSX Container Plug-in (NCP) 需要在主节点和 Kubernetes 节点上安装组件。

本章讨论了以下主题：

安装 NSX-T Data Center CNI 插件

NSX-T Data Center CNI 插件必须安装在 Kubernetes 节点上。

对于 Ubuntu，安装 NSX-T CNI 插件会将 AppArmor 配置文件 `ncp-apparmor` 复制到 `/etc/apparmor.d`，并加载它。安装之前，AppArmor 服务必须运行且目录 `/etc/apparmor.d` 必须存在。否则，安装将失败。可以使用以下命令检查 AppArmor 模块是否已启用：

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

可以使用以下命令检查 AppArmor 服务是否已启动：

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

如果安装 NSX-T CNI 插件时 AppArmor 服务未运行，安装完成时将显示以下消息：

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

该消息指示除了加载 AppArmor 配置文件之外，所有安装步骤都已完成。

ncp-apparmor 配置文件为 NSX 节点代理（称为 node-agent-apparmor）提供 AppArmor 配置文件，这与 docker-default 配置文件在以下方面有所不同：

- 移除了 deny mount 规则。
- 添加了 mount 规则。
- 添加了一些 network、capability、file 和 umount 选项。

您可以使用不同的配置文件替换 node-agent-apparmor 配置文件。但是，安装 NSX 节点代理时使用的文件 nsx-node-agent-ds.yml 中引用了配置文件名称 node-agent-apparmor。如果您使用其他配置文件，则必须在 nsx-node-agent-ds.yml 的 spec:template:metadata:annotations 部分下的以下条目中，指定配置文件名称：

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

步骤

1 下载适用于您的 Linux 发布版本的安装文件。

文件名是 nsx-cni-1.0.0.0.0.xxxxxxx-1.x86_64.rpm 或 nsx-cni-1.0.0.0.0.xxxxxxx.deb，其中 *xxxxxxx* 是内部版本号。

2 安装在步骤 1 中下载的 rpm 或 deb 文件。

该插件安装在 /opt/cni/bin 中。CNI 配置文件 10.nsx.conf 将复制到 /etc/cni/net.d 中。该 rpm 还会为环回插件安装 /etc/cni/net.d/99-loopback.conf 配置文件。

安装和配置 OVS

在工作节点上安装和配置 OVS (Open vSwitch)。

步骤

1 为您的 Linux 发布版本下载安装文件。

文件名为 openvswitch-common_2.10.x.xxxxxxx-1_amd64.deb、openvswitch-datapath-dkms_2.10.x.xxxxxxx-1_all.deb 和 openvswitch-switch_2.10.x.xxxxxxx-1_amd64.deb，其中 *xxxxxxx* 是内部版本号。

2 安装在步骤 1 中下载的 deb 文件。

3 对于 Ubuntu，请运行以下命令以重新加载 OVS 内核模块。

```
# systemctl force-reload openvswitch-switch
```

4 确保 OVS 正在运行。

```
# systemctl status openvswitch-switch.service
```

5 如果尚未创建 *br-int* 实例，请创建该实例。

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

6 将连接到节点逻辑交换机的网络接口 (*node-if*) 添加到 *br-int* 中。

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

运行以下命令以查看 `ofport` 的值，因为在 `ofport` 1 不可用时，OVS 将分配可用的端口。

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

如果 `ofport` 不是 1，请在 NSX 节点代理 DaemonSet yaml 文件的 `nsx_kube_proxy` 部分中相应地设置 `ovs_uplink_port` 选项。

7 确保 *br-int* 和 *node-if link* 状态为 up。

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

8 更新网络配置文件，以确保网络接口在重新引导后处于已启动状态。

对于 Ubuntu，请更新 `/etc/network/interfaces` 并添加以下行：

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

对于 RHEL，请更新 `/etc/sysconfig/network-scripts/ifcfg-<node-if>` 并添加以下行：

```
ONBOOT=yes
```

为 Kubernetes 节点配置 NSX-T Data Center 网络

本节介绍了如何为 Kubernetes 主节点和工作线程节点配置 NSX-T Data Center 网络。

每个节点必须具有至少两个网络接口。第一个接口是管理接口，可能位于 NSX-T Data Center Fabric 上，也可能不位于该 Fabric 上。另一个接口为 pod 提供网络，位于 NSX-T Data Center Fabric 上，并连接到称为节点逻辑交换机的逻辑交换机。管理和 pod IP 地址必须可路由，Kubernetes 运行状况检查才能正常工作。对于管理接口和 pod 之间的通信，NCP 自动创建一个 DFW 规则以允许运行状况检查和其他管理流量。您可以在 NSX Manager GUI 中查看该规则的详细信息。不应更改或删除该规则。

对于每个节点虚拟机，请确保为容器网络指定的 vNIC 连接到节点逻辑交换机。

NSX Container Plug-in (NCP) 必须知道用于每个节点中的容器流量的 vNIC 的 VIF ID。相应的逻辑交换机端口必须具有以下两个标记。对于其中一个标记，请指定节点的名称。对于另一个标记，请指定集群的名称。对于范围，请指定如下所示的相应值。

标记	范围
节点名称	ncp/node_name
集群名称	ncp/cluster

您可以从 NSX Manager GUI 中导航到**清单 > 虚拟机**以指定节点虚拟机的逻辑交换机端口。

如果 Kubernetes 节点名称发生变化，您必须更新 ncp/node_name 标记并重新启动 NCP。您可以使用以下命令获取节点名称：

```
kubectl get nodes
```

如果在运行 NCP 时将节点添加到集群中，您必须在运行 kubeadm join 命令之前将标记添加到逻辑交换机端口中。否则，新节点将没有网络连接。如果标记不正确或丢失，您可以执行以下步骤以解决该问题：

- 将正确的标记应用于逻辑交换机端口。
- 重新启动 NCP。

安装 NSX 节点代理

NSX 节点代理是一个 DaemonSet，每个 pod 将在其中运行两个容器。一个容器运行 NSX 节点代理，其主要职责是管理容器网络接口。它与 CNI 插件和 Kubernetes API 服务器进行交互。另一个容器运行 NSX Kube 代理，其唯一的职责是将群集 IP 转换为 pod IP 以实施 Kubernetes 服务抽象。它实施与上游 Kube 代理相同的功能。

步骤

1 下载 NCP Docker 映像。

文件名是 nsx-ncp-xxxxxxx.tar，其中 *xxxxxxx* 是内部版本号。

2 下载 NSX 节点代理 DaemonSet yaml 模板。

文件名是 nsx-node-agent-ds.yml。您可以编辑该文件，或者将其作为您自己的模板文件的示例。

3 将 NCP Docker 映像加载到您的映像注册表中。

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

4 编辑 nsx-node-agent-ds.yml。

将映像名称更改为加载的映像。

对于 Ubuntu，该 yaml 文件假定已启用 AppArmor。要查看是否启用了 AppArmor，请检查 /sys/module/apparmor/parameters/enabled 文件。如果未启用 AppArmor，请进行以下更改：

- 删除或取消注释以下行：

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

- 在 `securityContext` 下面，为 nsx-node-agent 容器和 nsx-kube-proxy 容器添加 `privileged:true` 行。例如：

```
securityContext:
    privileged:true
```

注 存在一个已知问题：如果在使用 hyperkube 映像的容器中运行 kubelet，kubelet 始终将 AppArmor 报告为已禁用，而无论实际处于何种状态。您必须对该 yaml 文件进行上述的相同更改。

注 在该 yaml 文件中，您必须指定为 `ncp.ini` 生成的 ConfigMap 必须挂载为只读卷。下载的 yaml 文件已具有该规范，不应对其进行更改。

5 使用以下命令创建 NSX 节点代理 DaemonSet。

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

nsx-node-agent-ds.yml 中的 ncp.ini 的 ConfigMap

示例 yaml 文件 `nsx-node-agent-ds.yml` 包含 NSX 节点代理的配置文件 `ncp.ini` 的 ConfigMap。该 ConfigMap 部分包含一些参数，您可以指定这些参数以自定义节点代理安装。

您下载的示例 `nsx-node-agent-ds.yml` 包含以下 `ncp.ini` 信息：

```
# 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

    # The log file path must be set to something like '/var/log/nsx-ujo/'. By
    # default, logging to file is disabled.
    #log_dir = None

    # Name of log file to send logging output to. If log_dir is set but log_file is
    # not, the binary name will be used, i.e., ncp.log, nsx_node_agent.log and
    # nsx_kube_proxy.log.
    #log_file = None
```

```
# max MB for each compressed file. Defaults to 100 MB
#log_rotation_file_max_mb = 100

# Total number of compressed backup files to store. Defaults to 5.
#log_rotation_backup_count = 5
[coe]
#
# Common options for Container Orchestrators
#

# Container orchestrator adaptor to plug in
# Options: kubernetes (default), openshift, pcf.
#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

# Once enabled, all projects in this cluster will be mapped to a NAT
# topology in NSX backend
#enable_snat = True

# The type of container node. Possible values are HOSTVM, BAREMETAL.
#node_type = HOSTVM

[ha]
#
# NCP High Availability configuration options
#

# Time duration in seconds of mastership timeout. NCP instance will
# remain master for this duration after elected. Note that the heartbeat
# period plus the update timeout must be less than this period. This
# is done to ensure that the master instance will either confirm
# liveness or fail before the timeout.
#master_timeout = 9

# Time in seconds between heartbeats for elected leader. Once an NCP
# instance is elected master, it will periodically confirm liveness based
# on this value.
#heartbeat_period = 3

# Timeout duration in seconds for update to election resource. If the
```

```
# update request does not complete before the timeout it will be
# aborted. Used for master heartbeats to ensure that the update finishes
# or is aborted before the master timeout occurs.
#update_timeout = 3

[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 OVS bridge to configure container interface.
#ovs_bridge = br-int

[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>
```

安装 NSX Container Plug-in

NSX Container Plug-in (NCP) 是作为 Docker 映像提供的。应在节点上运行 NCP 以提供基础架构服务。不建议在主节点上运行 NCP。

步骤

1 下载 NCP Docker 映像。

文件名是 `nsx-ncp-xxxxxxx.tar`，其中 *xxxxxxx* 是内部版本号。

2 下载 NCP ReplicationController yaml 模板。

文件名是 `ncp-rc.yml`。您可以编辑该文件，或者将其作为您自己的模板文件的示例。

3 将 NCP Docker 映像加载到您的映像注册表中。

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

4 （可选） 为 NSXError 对象的自定义资源定义下载 yaml 模板。

文件名为 `nsx-error-crd.yaml`。

5 （可选） 创建自定义资源。

```
kubectl create -f nsx-error-crd.yaml
```

6 编辑 `ncp-rc.yml`。

将映像名称更改为加载的映像。

指定 `nsx_api_managers` 参数。可以指定单个 NSX Manager 的 IP 地址、NSX Manager 群集中三个 NSX Manager 的 IP 地址（以逗号分隔）或 NSX Manager 群集的虚拟 IP 地址。例如：

```
    nsx_api_managers = 192.168.1.180
or
    nsx_api_managers = 192.168.1.181,192.168.1.182,192.168.1.183
```

（可选）在 [nsx_v3] 部分中指定 ca_file 参数。该值应该是在验证 NSX Manager 服务器证书时使用的 CA 包文件。如果未设置，将使用系统根 CA。如果为 nsx_api_managers 指定一个 IP 地址，则指定一个 CA 文件。如果为 nsx_api_managers 指定三个 IP 地址，则可以指定一个或三个 CA 文件。如果指定一个 CA 文件，则该文件将用于所有三个管理器。如果指定三个 CA 文件，则每个文件将用于 nsx_api_managers 中对应的 IP 地址。例如，

```
    ca_file = ca_file_for_all_mgrs
or
    ca_file = ca_file_for_mgr1,ca_file_for_mgr2,ca_file_for_mgr3
```

指定 nsx_api_cert_file 和 nsx_api_private_key_file 参数以使用 NSX-T Data Center 进行身份验证。

nsx_api_cert_file 是 PEM 格式的客户端证书文件的完整路径。该文件的内容应如下所示：

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

nsx_api_private_key_file 是 PEM 格式的客户端私钥文件的完整路径。该文件的内容应如下所示：

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

如果 Ingress 控制器配置为在 NAT 模式下运行，请指定 ingress_mode = nat 参数。

默认情况下，子网前缀 24 用于从 pod 逻辑交换机的 IP 块中分配的所有子网。要使用不同的子网大小，请在 [nsx_v3] 部分中更新 subnet_prefix 选项。

默认情况下，将启用 HA（高可用性）。可以使用以下规范禁用 HA：

```
[ha]
enable = False
```

（可选）在 ncp.ini 中启用 NSXError 报告错误。默认情况下，将禁用此设置。

```
[nsx_v3]
enable_nsx_err_crd = True
```

注 在该 yaml 文件中，您必须指定为 ncp.ini 生成的 ConfigMap 应挂载为只读卷。下载的 yaml 文件已具有该规范，不应对其进行更改。

7 创建 NCP ReplicationController。

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

结果

注 NCP 打开与 Kubernetes API 服务器的持续 HTTP 连接，以监视 Kubernetes 资源的生命周期事件。如果 API 服务器故障或网络故障导致 NCP 的 TCP 连接失效，则必须重新启动 NCP，以便其可以重新建立与 API 服务器的连接。否则，NCP 将错过新事件。

滚动更新 NCP ReplicationController 期间，在以下情况下，滚动更新完成后可能会看到两个 NCP pod 正在运行：

- 滚动更新期间重新引导容器主机。
- 由于新的映像不存在于 Kubernetes 节点上，滚动更新最初失败。下载映像，然后重新运行滚动更新后成功。

如果看到两个 NCP pod 正在运行，请执行以下操作：

- 删除其中一个 NCP pod。具体删除哪一个无关紧要。例如，

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

- 删除 NCP ReplicationController。例如，

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

ncp-rc.yml 中的 ncp.ini 的 ConfigMap

示例 yaml 文件 `ncp–rc.yml` 包含配置文件 `ncp.ini` 的 ConfigMap。该 ConfigMap 部分包含一些参数，您必须在安装 NCP 之前指定这些参数，如上一节中所述。

您下载的示例 `ncp–rc.yml` 包含以下 `ncp.ini` 信息：

```
# 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
```

```
# The log file path must be set to something like '/var/log/nsx-ujo/'. By
# default, logging to file is disabled.
#log_dir = None

# Name of log file to send logging output to. If log_dir is set but log_file is
# not, the binary name will be used, i.e., ncp.log, nsx_node_agent.log and
# nsx_kube_proxy.log.
#log_file = None

# max MB for each compressed file. Defaults to 100 MB
#log_rotation_file_max_mb = 100

# Total number of compressed backup files to store. Defaults to 5.
#log_rotation_backup_count = 5
[coe]
#
# Common options for Container Orchestrators
#

# Container orchestrator adaptor to plug in
# Options: kubernetes (default), openshift, pcf.
#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

# Once enabled, all projects in this cluster will be mapped to a NAT
# topology in NSX backend
#enable_snat = True

# The type of container node. Possible values are HOSTVM, BAREMETAL.
#node_type = HOSTVM

[ha]
#
# NCP High Availability configuration options
#

# Time duration in seconds of mastership timeout. NCP instance will
# remain master for this duration after elected. Note that the heartbeat
# period plus the update timeout must be less than this period. This
# is done to ensure that the master instance will either confirm
```

```
# liveness or fail before the timeout.
#master_timeout = 9

# Time in seconds between heartbeats for elected leader. Once an NCP
# instance is elected master, it will periodically confirm liveness based
# on this value.
#heartbeat_period = 3

# Timeout duration in seconds for update to election resource. If the
# update request does not complete before the timeout it will be
# aborted. Used for master heartbeats to ensure that the update finishes
# or is aborted before the master timeout occurs.
#update_timeout = 3

[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>

# If true, the NSX Manager server certificate is not verified. If false the CA
# bundle specified via "ca_file" will be used or if unsest the default system
# root CAs will be used. (boolean value)
#insecure = False

# Specify one or a list of CA bundle files 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. (list value)
#ca_file = <None>

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

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

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

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

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

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

# Number of times a HTTP redirect should be followed. (integer value)
```

```
#redirects = 2

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

# Subnet prefix of IP block. IP block will be retrieved from NSX API and
# recognised by tag 'cluster'.
# Prefix should be less than 31, as two addresses(the first and last addresses)
# need to be network address and broadcast address.
# The prefix is fixed after the first subnet is created. It can be changed only
# if there is no subnets in IP block.
#subnet_prefix = 24

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

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

# Used when ingress class annotation is missing
# if set to true, the ingress will be handled by nsx lbs
# otherwise will be handled by 3rd party ingress controller (e.g. nginx)
#default_ingress_class_nsx = True

# Path to the default certificate file for HTTPS load balancing
#lb_default_cert_path = <None>

# Path to the private key file for default certificate for HTTPS load balancing
#lb_priv_key_path = <None>

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

# Choice of persistence type for ingress traffic through L7 Loadbalancer.
# Accepted values:
# 'cookie'
# 'source_ip'
#l7_persistence = <None>

# Choice of persistence type for ingress traffic through L4 Loadbalancer.
# Accepted values:
# 'source_ip'
#l4_persistence = <None>
```

```
# Name or UUID of the tier0 router that project tier1 routers connect to
#tier0_router = <None>

# Name or UUID of the NSX overlay transport zone that will be used for creating
# logical switches for container networking. It must refer to an existing
# transport zone on NSX and every hypervisor that hosts the Kubernetes
# node VMs must join this transport zone
#overlay_tz = <None>

# Name or UUID of the NSX lb service that can be attached by virtual servers
#lb_service = <None>

# Name or UUID of the container ip blocks that will be used for creating
# subnets. If name, it must be unique
#container_ip_blocks = <None>

# Name or UUID of the container ip blocks that will be used for creating
# subnets for no-SNAT projects. If specified, no-SNAT projects will use these
# ip blocks ONLY. Otherwise they will use container_ip_blocks
#no_snat_ip_blocks = <None>

# Name or UUID of the external ip pools that will be used for allocating IP
# addresses which will be used for translating container IPs via SNAT rules
#external_ip_pools = <None>

# Name or UUID of the external ip pools that will be used for allocating IP
# addresses for exposing LoadBalancer type service and ingress
#external_ip_pools_lb = <None>

# Firewall sections for this cluster will be created below this mark section
#top_firewall_section_marker = <None>

# Firewall sections for this cluster will be created above this mark section
#bottom_firewall_section_marker = <None>

# Option to enabling error reporting through NSXError CRD
#enable_nsx_err_crd = False

# Option for user to define the maximum allowed virtual servers to be created
# for Service of type LoadBalancer in the cluster. The value should be an
# integer greater than zero.
# max_allowed_virtual_servers = <1000>
```

在 NCP pod 中挂载 PEM 编码的证书和私钥

如果具有 PEM 编码的证书和私钥，您可以更新 yaml 文件中的 NCP pod 定义以在 NCP pod 中挂载 TLS 密钥。

1 为证书和私钥创建一个 TLS 密钥。

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

2 更新 NCP pod 规范 yaml 以在 NCP pod 规范中将该密钥挂载为文件。

```
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 在 yaml 文件中更新 nsx_v3 选项 `nsx_api_cert_file` 和 `nsx_api_private_key_file`。

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

在 NCP pod 中挂载证书文件

如果在节点文件系统中具有一个证书文件，您可以更新 NCP pod 规范以在 NCP pod 中挂载该文件。例如，

```
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>
```

配置 syslog

您可以在容器中运行 syslog 代理（如 rsyslog 或 syslog-ng），以将日志从 NCP 和相关的组件发送到 syslog 服务器。

建议使用以下方法。有关 Kubernetes 中的日志记录的详细信息，请参阅 https://kubernetes.io/docs/concepts/cluster-administration/logging。

- 创建一个在 NCP 或 nsx-node-agent pod 中运行的 sidecar 容器。
- 在每个节点上运行一个 DaemonSet 副本。

注 在使用 sidecar 容器方法时，无法将 NSX CNI 插件日志发送到 syslog 服务器，因为该插件不在 pod 中运行。

为 syslog 创建 sidecar 容器

您可以为 syslog 配置一个 sidecar 容器，以在与 NCP 相同的 pod 中运行。以下过程假设 syslog 代理映像为 example/rsyslog。

步骤

1 配置 NCP 和 NSX 节点代理以记录到一个文件中。

在 NCP 和 NSX 节点代理的 yaml 文件中，设置 log_dir 参数并指定要挂载的卷。例如，

```
[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
```

您可以设置 `log_file` 参数以更改日志文件名称。默认名称为 `ncp.log`、`nsx_node_agent.log` 和 `nsx_kube_proxy.log`。如果 `log_dir` 选项设置为 `/var/log/nsx-ujo` 以外的路径，必须创建一个 hostPath 卷或 emptyDir 卷并挂载到相应的 pod 规范。

2 确保主机路径存在且可由用户 nsx-ncp 写入。

a 运行以下命令。

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

b 添加用户 nsx-ncp 或将主机路径模式更改为 777。

```
    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 在 NCP pod 的规范 yaml 文件中，为 syslog 添加一个 ConfigMap。例如，

```
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 在 NCP pod 的 yaml 文件中，添加 rsyslog 容器并挂载相应的卷，rsyslog 可以在其中查找配置数据并从其他容器中读取日志。例如，

```
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>
```

为 syslog 创建 DaemonSet 副本

可以使用该方法重定向所有 NCP 组件的日志。需要将应用程序配置为记录到 stderr（默认启用）。以下过程假设 syslog 代理映像为 example/rsyslog。

步骤

1 创建 DaemonSet yaml 文件。例如，

```
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 创建 DaemonSet。

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

示例：配置在 Sidecar 容器中运行的日志轮换和 Syslog

以下过程说明如何配置在 sidecar 容器中运行的日志轮换和 syslog。

创建日志目录和配置日志轮换

- 在所有节点（包括主节点）上创建日志目录，并将其所有者更改为 ID 为 1000 的任何用户。

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

- 在 /var/log/nsx-ujo 目录的所有节点上配置日志轮换。

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

创建 NCP 复制控制器

- 为 NCP 创建 ncp.ini 文件。

```
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
```

- 基于 ini 文件创建配置映射。

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

- 创建 NCP rsyslog 配置。

```
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
```

- 基于上述情况创建配置映射。

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

- 使用 rsyslog sidecar 创建 NCP 复制控制器。

```
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
```

- 使用上面的规范创建 NCP。

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

创建 NSX 节点代理 DaemonSet

- 为节点代理创建 rsyslog 配置。

```
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
```

- 基于上述情况创建 configmap。

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

- 使用 configmap sidecar 创建 DaemonSet。

```
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
```

- 创建 DaemonSet。

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

安全注意事项

在部署 NCP 时，请务必采取措施以保护 Kubernetes 和 NSX-T Data Center 环境。

将 NCP 限制为仅在指定的节点上运行

NCP 有权访问 NSX-T Data Center 管理层面，应将其限制为仅在指定的基础架构节点上运行。您可以使用相应的标签指定这些节点。然后，应将该标签的 nodeSelector 应用于 NCP ReplicationController 规范。例如，

```
  nodeSelector:
      nsx-infra: True
```

也可以使用其他机制（如关联性）以将 pod 分配给节点。有关详细信息，请参见 https://kubernetes.io/docs/concepts/configuration/assign-pod-node。

确保 Docker 引擎是最新的

Docker 定期发布安全更新。应实施一个自动化过程以应用这些更新。

禁止不受信任的容器的 NET_ADMIN 和 NET_RAW 功能

攻击者可能会利用 Linux 功能 NET_ADMIN 和 NET_RAW 以破坏 pod 网络。您应该禁用不受信任的容器的这两个功能。默认情况下，不会为非特权容器授予 NET_ADMIN 功能。如果 pod 规范明确启用该功能，或者将容器设置为特权模式，请务必小心。此外，对于不受信任的容器，请在容器规范的 SecurityContext 配置上的弃用的功能列表中指定 NET_RAW 以禁用 NET_RAW。例如，

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

基于角色的访问控制

Kubernetes 使用基于角色的访问控制 (RBAC) API 帮助作出授权决定，从而允许管理员动态配置策略。有关详细信息，请参见 https://kubernetes.io/docs/admin/authorization/rbac。

通常，群集管理员是唯一具有访问特权和角色的用户。对于用户和服务帐户，必须在授予访问权限时遵循最小特权原则。

建议使用以下准则：

- 将对 Kubernetes API 令牌的访问限制为需要它们的 pod。
- 将对 NCP ConfigMap 和 NSX API 客户端证书的 TLS 密钥的访问限制为 NCP pod。
- 阻止不需要访问 Kubernetes 网络 API 的 pod 进行该类访问。
- 添加 Kubernetes RBAC 策略以指定哪些 pod 有权访问 Kubernetes API。

NCP pod 的建议 RBAC 策略

使用 ServiceAccount 创建 NCP pod，并为该帐户授予一组最小特权。此外，不允许其他 pod 或 ReplicationController 访问 ConfigMap 和 TLS 密钥，它们是作为 NCP ReplicationController 和 NSX 节点代理的卷挂载的。

以下示例说明了如何为 NCP 指定角色和角色绑定：

```
# 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 有权访问 Kubernetes API 服务器的任何 pod 都可以访问使用 Kubernetes API 为 NSX-T Data Center 客户端证书和私钥对创建的 TLS 密钥。同样，在不使用服务帐户创建 pod 时，将在相同的命名空间中自动为其分配默认服务帐户，这会自动挂载令牌以访问 Kubernetes API。因此，必须将对这些令牌的访问限制为需要它们的 pod。

有关配置网络资源的提示

配置某些网络资源时，应该注意一些限制。

NSX-T Data Center 标记限制

标记对象时，NSX-T Data Center 有以下限制：

- 范围具有 128 个字符的限制。
- 标记具有 256 个字符的限制。
- 每个对象最多可以具有 30 个标记。

当 Kubernetes 或 OpenShift 注释被复制到 NSX-T Data Center 范围和标记，并且超出限制时，这些限制可能会导致出现问题。例如，如果某个标记适用于交换机端口，而该标记被用于防火墙规则，则可能无法按预期方式应用该规则，因为注释键或值在复制到范围或标记时被截断。

配置网络策略

网络策略使用标签选择器选择 pod 或命名空间。

NCP 的网络策略支持与 Kubernetes 所提供的支持一样，并且取决于 Kubernetes 版本。

- Kubernetes 1.11 - 可以指定以下规则选择器：
  - `podSelector`：可选择创建网络策略的命名空间中的所有 pod。

  - `namespaceSelector`：可选择所有命名空间。
  - `podSelector` 和 `namespaceSelector`：可选择 `namespaceSelector` 选定的命名空间中的所有 pod。
  - `ipBlockSelector`：如果 `ipBlockSelector` 与 `namespaceSelector` 或 `podSelector` 结合使用，则网络策略无效。`ipBlockSelector` 必须单独存在于策略规范中。
- Kubernetes 1.10 - 网络策略中的规则子句最多只能包含 `namespaceSelector`、`podSelector` 和 `ipBlock` 中的一个选择器。

Kubernetes API 服务器不对网络策略规范进行验证。可以创建无效的网络策略。NCP 将拒绝此类网络策略。如果更新网络策略以使其有效，NCP 仍不会处理该网络策略。必须删除该网络策略，然后使用有效的规范重新创建一个。

在 Pivotal Cloud Foundry 环境中安装 NCP

4

Pivotal Cloud Foundry (PCF) 是开源的平台即服务 (platform-as-a-service, PaaS) 提供程序。可以在 PCF 环境中安装 NSX Container Plug-in 以提供网络服务。

通过 Pivotal Ops Manager 创建的虚拟机必须具有到容器网络的第 3 层连接才能访问 NSX-T 功能。

高可用性 (High availability, HA) 会自动启用。

注　对安全组进行更改时，必须重新转储应用该安全组的所有应用程序。发生这种情况的原因可能是安全组应用于运行应用程序的空间，或者安全组是全局安全组。

本章讨论了以下主题：

在 Pivotal Cloud Foundry 环境中安装 NCP

NCP 通过 Pivotal Ops Manager 图形用户界面进行安装。

前提条件

全新安装 Pivotal Ops Manager、NSX-T Data Center 和 Pivotal Application Service (PAS)。请确保先安装 Ops Manager，然后再安装 NSX-T Data Center，最后安装 PAS。有关详细信息，请参见 Pivotal Cloud Foundry 文档。

步骤

1. 下载 PCF 的 NCP 安装文件。

   文件名称为 `VMware-NSX-T.<version>.<build>.pivotal`。
2. 以管理员身份登录到 Pivotal Ops Manager。
3. 单击**导入产品**。
4. 选择已下载的文件。
5. 单击 **Ops Manager Director for VMware vSphere** 图标。
6. 在 **vCenter 配置**的**设置**选项卡中，选择 **NSX 网络**，对于 **NSX 模式**，选择 **NSX-T**。
7. 在 **NSX CA 证书**字段中，提供 PEM 格式的证书。

8 单击**保存**。

9 单击左上角的**安装仪表板**以返回到仪表板。

10 单击 **Pivotal Application Service** 图标。

11 在**设置**选项卡中，选择导航窗格中的**网络**。

12 在**容器网络接口插件**下，选择**外部**。

13 单击左上角的**安装仪表板**以返回到仪表板。

14 单击**保存**。

15 单击左上角的**安装仪表板**以返回到仪表板。

16 单击 **VMware NSX-T** 图标。

17 输入 NSX Manager 的地址。

18 选择 NSX Manager 身份验证方法。

选项	操作
客户端证书身份验证	提供 NSX Manager 的证书和私钥。
使用用户名和密码进行基本身份验证	提供 NSX Manager 管理员用户名和密码。

19 在 **NSX Manager CA 证书**字段中，提供证书。

20 单击**保存**。

21 在导航窗格中选择 **NCP**。

22 输入 **PAS Foundation 名称**。

此字符串唯一地标识 NSX API 中的 PAS foundation。此字符串还用作 NCP 为 PAS foundation 创建的 NSX 资源的名称中的前缀。

23 输入**覆盖网络传输区域**。

24 输入 **Tier-0 路由器**。

25 指定一个或多个**容器网络的 IP 块**。

a 单击**添加**。

b 输入 **IP 块名称**。可以是新的 IP 块，也可以是现有的 IP 块。

c 仅适用于新 IP 块：以 CIDR 格式指定块，例如 10.1.0.0/16。

26 指定容器网络的子网前缀。

27 单击**为容器网络启用 SNAT** 以启用 SNAT。

28 指定一个或多个**用于向组织网络提供外部 (NAT) IP 地址的 IP 池**。

a 单击**添加**。

b 输入 **IP 池名称**。可以是新的 IP 池，也可以是现有的 IP 池。

c 仅适用于新 IP 池：通过提供 CIDR 和 IP 范围指定 IP 地址。

29 （可选） 输入**顶部防火墙区域标记**。

30 （可选） 输入**底部防火墙区域标记**。

31 （可选） 启用或禁用以下选项。

选项	默认值
记录丢弃的应用程序流量	已禁用。如果启用，则将记录由于防火墙规则而丢弃的流量。
为 NCP 日志记录启用调试级别	已启用。

32 单击**保存**。

33 （可选） 在导航窗格中选择 **NSX 节点代理**。

a 选中**为 NSX 节点代理启用调试级别的日志记录**以启用调试级别日志记录。

b 单击**保存**。

34 单击左上角的**安装仪表板**以返回到仪表板。

35 单击**应用更改**。

负载均衡

5

NSX-T Data Center 负载均衡器已与 Kubernetes 集成。

本章讨论了以下主题：

配置负载均衡

配置负载均衡涉及配置 Kubernetes LoadBalancer 服务或 Ingress 资源以及 NCP 复制控制器。

可以通过配置类型为 LoadBalancer 的 Kubernetes 服务创建第 4 层负载均衡器，通过配置 Kubernetes Ingress 资源创建第 7 层负载均衡器。

要在 NCP 中配置负载均衡，请在 `ncp-rc.yml` 文件中：

1 设置 `use_native_loadbalancer` = **`True`**。

2 （可选）将 `pool_algorithm` 设置为 **`ROUND_ROBIN`** 或 **`LEAST_CONNECTION/IP_HASH`**。默认值为 **`ROUND_ROBIN`**。

3 （可选）将 `service_size` 设置为 **`SMALL`**、**`MEDIUM`** 或 **`LARGE`**。默认值为 **`SMALL`**。

`LEAST_CONNECTION/IP_HASH` 算法意味着来自同一源 IP 地址的流量将被发送到相同的后端 pod。

有关不同大小的 NSX-T 负载均衡器支持内容的详细信息，请参见《NSX-T Data Center 管理指南》。

创建负载均衡器后，无法通过更新配置文件来更改负载均衡器大小，但是可以通过 NSX Manager UI 或 API 进行更改。

设置第 4 层和第 7 层负载均衡的持久性

可以在 NCP ConfigMap 中使用参数 `l4_persistence` 和 `l7_persistence` 指定持久性设置。可用于设置第 4 层持久性的选项为源 IP。可用于设置第 7 层持久性的选项为 cookie 和源 IP。默认值为 <None>。例如，

```
# Choice of persistence type for ingress traffic through L7 Loadbalancer.
# Accepted values:
# 'cookie'
# 'source_ip'
l7_persistence = cookie

# Choice of persistence type for ingress traffic through L4 Loadbalancer.
# Accepted values:
# 'source_ip'
l4_persistence = source_ip
```

对于 Kubernetes LoadBalancer 服务，如果禁用了全局第 4 层持久性（即 `l4_persistence` 设置为 `<None>`），您还可以在服务规范上指定 `sessionAffinity` 以配置服务持久性行为。如果 `l4_persistence` 设置为 `source_ip`，则可以使用服务规范上的 `sessionAffinity` 自定义服务持久性超时。默认第 4 层持久性超时为 10800 秒，与 Kubernetes 文档 (https://kubernetes.io/docs/concepts/services-networking/service) 中指定的服务超时相同。具有默认持久性超时的所有服务将使用相同的 NSX-T 负载均衡器持久性配置文件。将为每个具有非默认持久性超时的服务创建专用的配置文件。

注 如果 Ingress 的后端服务的服务类型为 LoadBalancer，则该服务的第 4 层虚拟服务器和 Ingress 的第 7 层虚拟服务器不能具有不同的持久性设置，例如，对于第 4 层，采用 `source_ip`，而对于第 7 层，采用 `cookie`。在这种场景下，这两个虚拟服务器的持久性设置必须相同（`source_ip`、`cookie` 或 `None`），或者其中一个为 `None`（另一个的设置可以为 `source_ip` 或 `cookie`）。下面列出了这种场景的一个示例：

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: cafe-ingress
spec:
  rules:
  - host: cafe.example.com
    http:
      paths:
      - path: /tea
        backend:
          serviceName: tea-svc
          servicePort: 80
-----
apiVersion: v1
kind: Service
metadata:
  name: tea-svc <==== same as the Ingress backend above
  labels:
    app: tea
spec:
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
    name: tcp
  selector:
    app: tea
  type: LoadBalancer
```

输入

- NSX-T Data Center 将分别为具有 TLS 规范和不具有 TLS 规范的 Ingress 创建一个第 7 层负载均衡器。
- 所有 Ingress 都将获得单个 IP 地址。

- 从 `ncp.ini` 的 `[nsx_v3]` 部分中的 `external_ip_pools` 选项指定的外部 IP 池为 Ingress 资源分配 IP 地址。将在此 IP 地址以及 HTTP 端口 80 和 HTTPS 端口 443 上公开负载均衡器。
- 从 `ncp.ini` 的 `[nsx_v3]` 部分中的 `external_ip_pools_lb` 选项指定的外部 IP 池为 Ingress 资源分配 IP 地址。如果 `external_ip_pools_lb` 选项不存在，则使用 `external_ip_pools` 指定的池。将在此 IP 地址以及 HTTP 端口 80 和 HTTPS 端口 443 上公开负载均衡器。
- 可以通过更改配置并重新启动 NCP 来更改为不同的 IP 池。
- 可以为 TLS 指定默认证书。有关生成证书并将证书挂载到 NCP pod 的信息，请参见下文。
- 将在 HTTP 虚拟服务器（端口 80）上托管不具有 TLS 规范的 Ingress。
- 将在 HTTPS 虚拟服务器（端口 443）上托管具有 TLS 规范的 Ingress。负载均衡器将充当 SSL 服务器并终止客户端 SSL 连接。
- 密钥和 Ingress 的创建顺序无关紧要。如果存在密钥对象且该对象正由某个 Ingress 引用，则将在 NSX-T Data Center 中导入证书。如果删除了密钥或删除了引用该密钥的最后一个 Ingress，则将删除与该密钥对应的证书。
- 支持通过添加或移除 TLS 部分来修改 Ingress。从 Ingress 规范中移除 `tls` 密钥后，Ingress 规则将从 HTTPS 虚拟服务器（端口 443）传输到 HTTP 虚拟服务器（端口 80）。同样，将 `tls` 密钥添加到 Ingress 规范后，Ingress 规则将从 HTTP 虚拟服务器（端口 80）传输到 HTTPS 虚拟服务器（端口 443）。
- 如果单个集群的 Ingress 定义中存在重复的规则，则只会应用第一个规则。
- 每个集群只支持一个具有默认后端的 Ingress。不匹配任何 Ingress 规则的流量将被转发到默认后端。
- 如果存在多个具有默认后端的 Ingress，则只会配置第一个输入。其他 Ingress 将被注释为出现错误。
- 使用正则表达式字符“.”和“*”支持 URI 通配符匹配。例如，路径“/coffee/.*”匹配“/coffee/”及后跟零个、一个或多个字符，如“/coffee/”、“/coffee/a”、“/coffee/b”，但不匹配“/coffee”、“/coffeecup”或“/coffeecup/a”。

  Ingress 规范示例：

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

- 您可以通过将注释添加到 Ingress 资源来配置 URL 请求重写。例如，

  ```
  kind: Ingress
  metadata:
    name: cafe-ingress
  ```

```
  annotations:
    ingress.kubernetes.io/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
```

路径 /tea 和 /coffee 将在 URL 发送到后端服务之前重写到 /。

- 支持 Ingress 注释 kubernetes.io/ingress.allow-http。
  - 如果注释设置为 **false**，将仅创建 HTTPS 规则。
  - 如果注释设置为 **true** 或缺失，将创建 HTTP 规则。此外，如果 Ingress 规范中存在 TLS 部分，则将创建 HTTPS 规则。
- 错误作为注释添加到 Ingress 资源中。错误键为 ncp/error.loadbalancer，警告键为 ncp/warning.loadbalancer。可能的错误和警告包括：
  - ncp/error.loadbalancer: DEFAULT_BACKEND_IN_USE

    该错误表示具有默认后端的 Ingress 已存在。Ingress 将处于非活动状态。无论是否使用 TLS，对于一组 Ingress 而言，只能有一个默认后端。要解决此错误，请删除 Ingress 并使用正确的规范重新创建。
  - ncp/warning.loadbalancer: SECRET_NOT_FOUND

    该错误表示 Ingress 规范中指定的密钥不存在。Ingress 将部分处于活动状态。要解决此错误，请创建缺少的密钥。请注意，警告出现在注释中后，不会在 Ingress 资源的生命周期内将其清除。
  - ncp/warning.loadbalancer: INVALID_INGRESS

    此错误表示以下条件之一成立。Ingress 将处于非活动状态。要解决此错误，请删除 Ingress 并使用正确的规范重新创建。
    - Ingress 规则与同一 Kubernetes 集群中的另一 Ingress 规则冲突。
    - allow-http 注释设置为 **False**，且 Ingress 没有 TLS 区域。
    - Ingress 规则未指定 host 和 path。此类 Ingress 规则具有与 Ingress 默认后端相同的功能。请改用 Ingress 默认后端。

类型为 LoadBalancer 的服务

- NSX-T Data Center 将为每个服务端口创建一个第 4 层负载均衡器虚拟服务器和池。
- TCP 和 UDP 均受支持。

- 每个服务都将有一个唯一的 IP 地址。
- 根据 LoadBalancer 定义中的 `loadBalancerIP` 字段，将从外部 IP 池中为服务分配一个 IP 地址。`loadBalancerIP` 字段可以为空，具有 IP 地址或具有 IP 池的名称或 ID。如果 `loadBalancerIP` 字段为空，则从 `ncp.ini` 的 `[nsx_v3]` 部分中的 `external_ip_pools_lb` 选项指定的外部 IP 池中分配 IP。如果 `external_ip_pools_lb` 选项不存在，则使用 `external_ip_pools` 指定的池。将在此 IP 地址和服务端口上公开 LoadBalancer 服务。
- 可以通过更改配置并重新启动 NCP 来更改为不同的 IP 池。
- `loadBalancerIP` 指定的 IP 池必须具有标记 `scope: ncp/owner, tag: cluster:<cluster_name>`。
- 错误作为注释添加到服务中。错误键为 `ncp/error.loadbalancer`。可能的错误包括：
  - `ncp/error.loadbalancer`: IP_POOL_NOT_FOUND

    该错误表示指定了 `loadBalancerIP`: `<nsx-ip-pool>`，但 `<nsx-ip-pool>` 不存在。服务将处于非活动状态。要解决此错误，请指定有效的 IP 池，然后删除并重新创建该服务。
  - `ncp/error.loadbalancer`: IP_POOL_EXHAUSTED

    该错误表示指定了 `loadBalancerIP`: `<nsx-ip-pool>`，但 IP 池已用尽其 IP 地址。服务将处于非活动状态。要解决此错误，请指定包含可用 IP 地址的 IP 池，然后删除并重新创建该服务。
  - `ncp/error.loadbalancer`: IP_POOL_NOT_UNIQUE

    此错误表示多个 IP 池具有 `loadBalancerIP`: `<nsx-ip-pool>` 指定的名称。服务将处于非活动状态。
  - `ncp/error.loadbalancer`: POOL_ACCESS_DENIED

    此错误表示 `loadBalancerIP` 指定的 IP 池不具有标记 `scope: ncp/owner, tag: cluster:<cluster_name>`，或标记中指定的集群与 Kubernetes 集群的名称不匹配。
  - `ncp/error.loadbalancer`: LB_VIP_CONFLICT

    此错误表示 `loadBalancerIP` 字段中的 IP 与活动服务的 IP 相同。服务将处于非活动状态。
- 支持自动缩放第 4 层负载均衡器。如果创建或修改 Kubernetes LoadBalancer 服务以便需要更多的虚拟服务器，而现有的第 4 层负载均衡器没有足够的容量，将创建新的第 4 层负载均衡器。NCP 也将删除不再连接虚拟服务器的第 4 层负载均衡器。默认情况下，将启用该功能。可以通过在 NCP ConfigMap 中将 `l4_lb_auto_scaling` 设置为 **`false`** 禁用此功能。

负载均衡器和网络策略

将流量从 NSX 负载均衡器虚拟服务器转发到 pod 时，源 IP 是 Tier-1 路由器上行链路端口的 IP 地址。此地址位于专用的 Tier-1 传输网络中，可能会导致基于 CIDR 的网络策略禁止本应允许的流量。要避免出现此问题，必须配置网络策略以使 Tier-1 路由器上行链路端口的 IP 地址包含在允许的 CIDR 块中。该内部 IP 地址将显示在 `status.loadbalancer.ingress.ip` 字段中，并在 Ingress 资源上显示为注释 (`ncp/internal_ip_for_policy`)。

例如，如果虚拟服务器的外部 IP 地址是 4.4.0.5，内部 Tier-1 路由器的上行链路端口的 IP 地址为 100.64.224.11，则状态为：

```
status:
  loadBalancer:
  ingress:
  - ip: 4.4.0.5
  - ip: 100.64.224.11
```

Ingress 资源和 LoadBalancer 类型服务资源上的注释为：

```
ncp/internal_ip_for_policy: 100.64.224.11
```

IP 地址 100.64.224.11 必须属于网络策略的 ipBlock 选择器中允许的 CIDR。例如，

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
...
ingress:
- from:
  - ipBlock:
     cidr: 100.64.224.11/32
```

用于生成 CA 签名证书的示例脚本

以下脚本可分别生成存储在文件 <filename>.crt 和 <finename>.key 中的 CA 签名证书和私钥。genrsa 命令可生成 CA 密钥。应对 CA 密钥进行加密。您可以使用 aes256 等命令指定加密方法。

```
#!/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
```

将默认证书和密钥挂载到 NCP Pod 中

生成证书和私钥后，请将其置于主机虚拟机上的目录 /etc/nsx-ujo 中。假设证书文件和密钥文件分别命名为 lb-default.crt 和 lb-default.key，请编辑 ncp-rc.yaml 以便主机上的这些文件挂载到 pod 中。例如，

```
spec:
  ...
  containers:
  - name: nsx-ncp
    ...
```

```
    volumeMounts:
    ...
    - name: lb-default-cert
      # Mount path must match nsx_v3 option "lb_default_cert_path"
      mountPath: /etc/nsx-ujo/lb-default.crt
    - name: lb-priv-key
      # Mount path must match nsx_v3 option "lb_priv_key_path"
      mountPath: /etc/nsx-ujo/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
```

第三方 Ingress 控制器

您可以将 NCP 配置为支持第三方 Ingress 控制器。

编辑 ncp.ini 文件

您必须编辑配置文件 /var/vcap/data/jobs/ncp/xxxxxxxx/config/ncp.ini（其中 xxxxxxxx 是 BOSH 部署 ID）。该文件随后将被复制到 rootfs，并在每次 NCP 重新启动时由 NCP 使用。该文件必须在每个主节点上进行编辑。

重要事项 对 ncp.ini 所做的更改不会跨 PKS 集群更新进行保留。如果通过 PKS 磁贴进行更改，然后更新 PKS 部署，则对 ncp.ini 所做的更改将会丢失。

相关选项位于 nsx_v3 部分中。

- use_native_loadbalancer - 如果设置为 **False**，NCP 将不会处理类型为 Loadbalancer 更新的任何 Ingress 或服务，而不论其注释为何。此设置适用于整个 PKS 集群。默认值为 **True**。
- default_ingress_class_nsx - 如果设置为 **True**，NCP 将成为默认 Ingress 控制器，并将处理带有 kubernetes.io/ingress.class: "nsx" 注释的 Ingress 和不带任何注释的 Ingress。如果设置为 **False**，NCP 将仅处理带有 kubernetes.io/ingress.class: "nsx" 注释的 Ingress。默认值为 **True**。

如果您希望 NCP 为 NGINX 控制器容器分配浮动 IP 并使用该浮动 IP 更新 Ingress 的状态，请执行以下操作：

- 在 ncp.ini 的 k8s 部分中，设置 ingress_mode=nat。
- 将注释 ncp/ingress-controller: "True" 添加到 NGINX Ingress 控制器容器中。

NCP 将使用 NGINX Ingress 控制器容器的浮动 IP 更新带有 kubernetes.io/ingress.class: "nginx" 注释的 Ingress 的状态。如果 default_ingress_class_nsx=False，NCP 还将使用 NGINX Ingress 控制器容器的浮动 IP 更新不带 kubernetes.io/ingress.class 注释的 Ingress 的状态。

注意：即使 NGINX Ingress 控制器容器不带 `ncp/ingress-controller: "True"` 注释，NCP 也会将上述 Ingress 的状态更新为 `loadBalancer: {}`。之后，Ingress 可能会陷入以下循环中：NGINX 控制器将 Ingress 状态更新为 `loadBalancer: {ingress: [{ip: <IP>}]}`，然后 NCP 又将 Ingress 状态更新为 `loadBalancer: {}`。为了避免出现这种情况，请执行以下步骤：

- 如果 Ingress 控制器来自 https://github.com/kubernetes/ingress-nginx，
  - 在 Ingress 控制器上，将 `ingress-class` 更改为 `"nginx"` 以外的内容。
  - 如果存在带有 `kubernetes.io/ingress-class: "nginx"` 注释的 Ingress，请将注释更改为不同的值。
  - 有关详细信息，请参见 https://kubernetes.github.io/ingress-nginx/user-guide/multiple-ingress。
- 如果 Ingress 控制器来自 https://github.com/nginxinc/kubernetes-ingress，
  - 在 Ingress 控制器上，将 `ingress-class` 更改为 `"nginx"` 以外的内容。
  - 如果存在带有 `kubernetes.io/ingress-class: "nginx"` 注释的 Ingress，请将注释更改为不同的值。
  - 在 Ingress 控制器容器上，将 `use-ingress-class-only` 设置为 **True**。这将阻止此控制器更新不带 `kubernetes.io/ingress-class` 注释的 Ingress。
  - 有关详细信息，请参见 https://github.com/kubernetes/ingress-nginx/blob/master/docs/user-guide/multiple-ingress.md。

场景 1：NCP 处理 Ingress，但不是默认 Ingress 控制器。

请按照以下过程操作，让 NCP 处理 `nsx` 类 Ingress。

1 在每个主节点上编辑 `ncp.ini` 的 `nsx_v3` 部分。

   a 将 `default_ingress_class_nsx` 设置为 **False**。

   b 保留 `use_native_loadbalancer` 设置为默认值 **True**。

2 在每个主节点上重新启动 NCP。这可能会导致发生主节点故障切换。

3 在希望 NCP 处理的所有 Ingress 中添加 `kubernetes.io/ingress.class: "nsx"` 注释。

场景 2：NCP 是默认 Ingress 控制器。

请按照以下过程操作：

1 无需编辑 `ncp.ini`，但需确保每个 Ingress 均已添加注释。

2 要由 NCP 处理的 Ingress 应添加 **kubernetes.io/ingress.class: "nsx"** 注释。

   尽管 NCP 将处理不带 `kubernetes.io/ingress.class` 注释的 Ingress，但在具有多个 Ingress 控制器的情况下，最佳做法是始终添加 `kubernetes.io/ingress.class` 注释，而不是依赖于默认 Ingress 控制器行为。

3 要由第三方 Ingress 控制器处理的 Ingress 必须使用这些 Ingress 控制器所需的值进行注释。

重要事项 除非目标是将 NGINX 设为默认 Ingress 控制器，否则请勿使用 **`nginx`** 作为 NGINX Ingress 控制器，因为这会使 NGINX 成为默认 Ingress 控制器。

场景 3：NCP 不处理任何 Ingress，而不论其注释为何。

请按照以下过程操作：

1 在每个主节点上编辑 `ncp.ini` 的 `nsx_v3` 部分。

   a 将 `use_native_loadbalancer` 设置为 **`False`**。`default_ingress_class_nsx` 的值现在无关紧要。

2 在每个主节点上重新启动 NCP。这可能会导致发生主节点故障切换。

请注意，NCP 也不会处理类型为 LoadBalancer 的服务

管理 NSX Container Plug-in

6

您可以从 NSX Manager GUI 或命令行界面 (Command Line Interface, CLI) 中管理 NSX Container Plug-in。

注 如果容器主机虚拟机在 ESXi 6.5 上运行，并且该虚拟机通过 vMotion 迁移到另一个 ESXi 6.5 主机，则运行在容器主机上的容器将断开与运行在其他容器主机上的容器的连接。您可以通过先断开然后重新连接容器主机的 vNIC 来解决该问题。ESXi 6.5 Update 1 或更高版本不会出现此问题。

Hyperbus 会在管理程序中保留 VLAN ID 4094 用于 PVLAN 配置，且不能更改此 ID。为避免任何 VLAN 冲突，请勿将 VLAN 逻辑交换机或 VTEP vmknic 配置为具有相同的 VLAN ID。

本章讨论了以下主题：

显示存储在 Kubernetes 资源 NSXError 中的错误信息

对于具有 NSX 后端故障的每个 Kubernetes 资源对象，会创建一个包含错误信息的 NSXError 对象。还有一个包含所有群集范围错误的错误对象。

默认情况下，不启用此功能。要启用，必须在安装 NCP 时在 `ncp.ini` 中将 `enable_nsx_err_crd` 设置为 True。

注 不得创建、更新或删除 NSXError 对象。

用于显示 NSXError 对象的命令：

- `kubectl get nsxerrors`

  列出所有 NSXError 对象。

- `kubectl get nsxerrors -l error-object-type=<type of resource>`

  列出与特定类型的 Kubernetes 对象相关的 NSXError 对象，例如类型为 `services` 的对象。

- `kubectl get nsxerrors <nsxerror name> –o yaml`

  显示 NSXError 对象的详细信息。

- `kubectl get svc <service name> –o yaml | grep nsxerror`

  查找与特定服务关联的 NSXError。

显示 NSXError 对象的详细信息时，规范部分包含以下重要信息。例如，

```
error–object–id: default.svc–1
error–object–name: svc–1
error–object–ns: default
error–object–type: services
message:
– '[2019–01–21 20:25:36]23705: Number of pool members requested exceed LoadBalancerlimit'
```

在此示例中，命名空间为 `default`。服务的名称为 `svc–1`。Kubernetes 资源的类型为 `services`。

在此版本中，NSXError 对象支持以下错误。

- 由于存在 NSX Edge 限制，自动缩放无法分配额外的负载平衡器。
- 负载平衡器虚拟服务器的数量超过限制（未启用自动缩放）。
- 负载平衡器服务器池的数量超过限制。
- 负载平衡器服务器池成员的数量超过负载平衡器限制或 NSX Edge 限制。
- 在处理 LoadBalancer 类型服务时，浮动 IP 地址已用尽。

CIF 连接的逻辑端口

CIF（容器接口）是容器上连接到交换机上的逻辑端口的网络接口。这些端口称为 CIF 连接的逻辑端口。

您可以从 NSX Manager GUI 中管理 CIF 连接的逻辑端口。

管理 CIF 连接的逻辑端口

导航到**网络 > 交换 > 端口**以查看所有逻辑端口，包括 CIF 连接的逻辑端口。单击 CIF 连接的逻辑端口的连接链接以查看连接信息。单击逻辑端口链接以打开包含四个选项卡的窗格：“概览”、“监控”、“管理”和“相关”。单击**相关 > 逻辑端口**将显示上行链路交换机上的相关逻辑端口。有关交换机端口的详细信息，请参阅《NSX-T 管理指南》。

网络监控工具

以下工具支持 CIF 连接的逻辑端口。有关这些工具的详细信息，请参阅《NSX-T 管理指南》。

- 流跟踪
- 端口连接
- IPFIX

- 支持使用连接到容器的逻辑交换机端口的 GRE 封装的远程端口镜像。有关详细信息，请参阅《NSX-T 管理指南》中的“了解端口镜像交换配置文件”。但是，不支持通过管理器 UI 执行 CIF 到 VIF 端口的端口镜像。

CLI 命令

要运行 CLI 命令，请登录到 NSX Container Plug-in 容器，然后打开一个终端并运行 `nsxcli` 命令。

也可以在节点上运行以下命令以显示 CLI 提示符：

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

表 6-1. 用于 NCP 容器的 CLI 命令

类型	命令	备注
状态	get ncp-master status	适用于 Kubernetes 和 PCF。
状态	get ncp-nsx status	适用于 Kubernetes 和 PCF。
状态	get ncp-watcher <watcher-name>	适用于 Kubernetes 和 PCF。
状态	get ncp-watchers	适用于 Kubernetes 和 PCF。
状态	get ncp-k8s-api-server status	仅适用于 Kubernetes。
状态	check projects	仅适用于 Kubernetes。
状态	check project <project-name>	仅适用于 Kubernetes。
状态	get ncp-bbs status	仅适用于 PCF。
状态	get ncp-capi status	仅适用于 PCF。
状态	get ncp-policy-server status	仅适用于 PCF。
缓存	get project-caches	仅适用于 Kubernetes。
缓存	get project-cache <project-name>	仅适用于 Kubernetes。
缓存	get namespace-caches	仅适用于 Kubernetes。
缓存	get namespace-cache <namespace-name>	仅适用于 Kubernetes。
缓存	get pod-caches	仅适用于 Kubernetes。
缓存	get pod-cache <pod-name>	仅适用于 Kubernetes。
缓存	get ingress-caches	仅适用于 Kubernetes。
缓存	get ingress-cache <ingress-name>	仅适用于 Kubernetes。
缓存	get ingress-controllers	仅适用于 Kubernetes。
缓存	get ingress-controller <ingress-controller-name>	仅适用于 Kubernetes。
缓存	get network-policy-caches	仅适用于 Kubernetes。

表 6-1. 用于 NCP 容器的 CLI 命令 （续）

类型	命令	备注
缓存	get network-policy-cache <pod-name>	仅适用于 Kubernetes。
缓存	get asg-caches	仅适用于 PCF。
缓存	get asg-cache <asg-ID>	仅适用于 PCF。
缓存	get org-caches	仅适用于 PCF。
缓存	get org-cache <org-ID>	仅适用于 PCF。
缓存	get space-caches	仅适用于 PCF。
缓存	get space-cache <space-ID>	仅适用于 PCF。
缓存	get app-caches	仅适用于 PCF。
缓存	get app-cache <app-ID>	仅适用于 PCF。
缓存	get instance-caches <app-ID>	仅适用于 PCF。
缓存	get instance-cache <app-ID> <instance-ID>	仅适用于 PCF。
缓存	get policy-caches	仅适用于 PCF。
支持	get ncp-log file <filename>	适用于 Kubernetes 和 PCF。
支持	get ncp-log-level	适用于 Kubernetes 和 PCF。
支持	set ncp-log-level <log-level>	适用于 Kubernetes 和 PCF。
支持	get support-bundle file <filename>	仅适用于 Kubernetes。
支持	get node-agent-log file <filename>	仅适用于 Kubernetes。
支持	get node-agent-log file <filename> <node-name>	仅适用于 Kubernetes。

表 6-2. 用于 NSX 节点代理容器的 CLI 命令

类型	命令
状态	get node-agent-hyperbus status
缓存	get container-cache <container-name>
缓存	get container-caches

表 6-3. 用于 NSX Kube 代理容器的 CLI 命令

类型	命令
状态	get ncp-k8s-api-server status
状态	get kube-proxy-watcher <watcher-name>

表 6-3. 用于 NSX Kube 代理容器的 CLI 命令 （续）

类型	命令
状态	get kube-proxy-watchers
状态	dump ovs-flows

用于 NCP 容器的状态命令

- 显示 NCP 主节点的状态

  ```
  get ncp-master status
  ```

  示例：

  ```
  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
  ```

- 显示 NCP 和 NSX Manager 之间的连接状态

  ```
  get ncp-nsx status
  ```

  示例：

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

- 显示 Ingress、命名空间、pod 和服务的监视程序状态

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

  示例：

  ```
  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
```

- 显示 NCP 和 Kubernetes API 服务器之间的连接状态

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

示例：

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

- 检查所有项目或特定项目

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

示例：

```
kubenode> 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
```

- 检查 NCP 和 PCF BBS 之间的连接状态

  ```
  get ncp-bbs status
  ```

  示例：

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

- 检查 NCP 和 PCF CAPI 之间的连接状态

  ```
  get ncp-capi status
  ```

  示例：

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

- 检查 NCP 和 PCF 策略服务器之间的连接状态

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

  示例：

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

用于 NCP 容器的缓存命令

- 获取项目或命名空间的内部缓存

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

  示例：

  ```
  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
```

- 获取 pod 的内部缓存

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

示例：

```
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
```

- 获取所有 Ingress 缓存或特定 Ingress 缓存

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

示例：

```
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


kubenode> 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
```

- 获取有关所有 Ingress 控制器或特定 Ingress 控制器的信息，包括已禁用的控制器

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

示例：

```
kubenode> 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
```

- 获取网络策略缓存或特定缓存

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

示例：

```
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
```

- 获取所有 ASG 缓存或特定 ASG 缓存

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

示例：

```
node> get asg-caches
    edc04715-d04c-4e63-abbc-db601a668db6:
        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-abbc-db601a668db6
    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:
```

- 获取所有组织缓存或特定组织缓存

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

示例：

```
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
```

- 获取所有空间缓存或特定空间缓存

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

示例：

```
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
```

- 获取所有应用程序缓存或特定应用程序缓存

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

示例：

```
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-bbff-a4badf21eb09
    space_id: 7870d134-7997-4373-b665-b6a910413c47
```

- 获取某个应用程序的所有实例缓存或特定实例缓存

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

示例：

```
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
```

- 获取所有策略缓存

```
get policy-caches
```

示例：

```
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
```

用于 NCP 容器的支持命令

- 在文件存储中保存 NCP 支持包

  支持包包含 pod 中的所有容器的日志文件并具有 **tier:nsx-networking** 标签。包文件采用 tgz 格式，并保存在 CLI 默认文件存储目录 /var/vmware/nsx/file-store 中。您可以使用 CLI file-store 命令将包文件复制到远程站点中。

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

  示例：

  ```
  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
  ```

- 在文件存储中保存 NCP 日志

  日志文件以 tgz 格式保存在 CLI 默认文件存储目录 /var/vmware/nsx/file-store 中。您可以使用 CLI file-store 命令将包文件复制到远程站点中。

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

  示例：

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

- 在文件存储中保存节点代理日志

  保存一个节点或所有节点的节点代理日志。日志以 tgz 格式保存在 CLI 默认文件存储目录 /var/vmware/nsx/file-store 中。您可以使用 CLI file-store 命令将包文件复制到远程站点中。

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

  示例：

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

- 获取并设置日志级别

  可用的日志级别为 NOTSET、DEBUG、INFO、WARNING、ERROR 和 CRITICAL。

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

  示例：

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

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

用于 NSX 节点代理容器的状态命令

- 显示节点代理和该节点上的 HyperBus 之间的连接状态。

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

  示例：

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

用于 NSX 节点代理容器的缓存命令

- 获取 NSX 节点代理容器的内部缓存。

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

  示例：

  ```
  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
  ```

用于 NSX Kube 代理容器的状态命令

- 显示 Kube 代理和 Kubernetes API 服务器之间的连接状态

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

  示例：

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

- 显示 Kube 代理监视程序状态

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

  示例：

  ```
  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
  ```

- 在节点上转储 OVS 流量

  ```
  dump ovs-flows
  ```

示例：

```
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
```

错误代码

本部分列出了各种组件生成的错误代码。

NCP 错误代码

错误代码	说明
NCP00001	配置无效
NCP00002	初始化失败
NCP00003	状态无效
NCP00004	适配器无效
NCP00005	找不到证书
NCP00006	找不到令牌
NCP00007	NSX 配置无效
NCP00008	NSX 标记无效
NCP00009	NSX 连接失败
NCP00010	找不到节点标记
NCP00011	节点逻辑交换机端口无效
NCP00012	父 VIF 更新失败
NCP00013	VLAN 用尽
NCP00014	VLAN 释放失败
NCP00015	IP 池用尽

错误代码	说明
NCP00016	IP 释放失败
NCP00017	IP 块用尽
NCP00018	IP 子网创建失败
NCP00019	IP 子网删除失败
NCP00020	IP 池创建失败
NCP00021	IP 池删除失败
NCP00022	逻辑路由器创建失败
NCP00023	逻辑路由器更新失败
NCP00024	逻辑路由器删除失败
NCP00025	逻辑交换机创建失败

错误代码	说明
NCP00026	逻辑交换机更新失败
NCP00027	逻辑交换机删除失败
NCP00028	逻辑路由器端口创建失败
NCP00029	逻辑路由器端口删除失败
NCP00030	逻辑交换机端口创建失败
NCP00031	逻辑交换机端口更新失败
NCP00032	逻辑交换机端口删除失败
NCP00033	找不到网络策略
NCP00034	防火墙创建失败
NCP00035	防火墙读取失败
NCP00036	防火墙更新失败
NCP00037	防火墙删除失败
NCP00038	找到多个防火墙
NCP00039	NS 组创建失败
NCP00040	NS 组删除失败
NCP00041	IP 集创建失败
NCP00042	IP 集更新失败

错误代码	说明
NCP00043	IP 集删除失败
NCP00044	SNAT 规则创建失败
NCP00045	SNAT 规则删除失败
NCP00046	适配器 API 连接失败
NCP00047	适配器监控程序异常
NCP00048	负载均衡器服务删除失败
NCP00049	负载均衡器虚拟服务器创建失败
NCP00050	负载均衡器虚拟服务器更新失败

错误代码	说明
NCP00051	负载均衡器虚拟服务器删除失败
NCP00052	负载均衡器池创建失败
NCP00053	负载均衡器池更新失败
NCP00054	负载均衡器池删除失败
NCP00055	负载均衡器规则创建失败
NCP00056	负载均衡器规则更新失败
NCP00057	负载均衡器规则删除失败
NCP00058	负载均衡器池 IP 释放失败
NCP00059	找不到负载均衡器虚拟服务器和服务关联
NCP00060	NS 组更新失败
NCP00061	防火墙规则获取失败
NCP00062	NS 组没有标准
NCP00063	找不到节点虚拟机
NCP00064	找不到节点 VIF
NCP00065	证书导入失败
NCP00066	证书取消导入失败
NCP00067	SSL 绑定更新失败
NCP00068	未找到 SSL 配置文件
NCP00069	找不到 IP 池

错误代码	说明
NCP00070	找不到 T0 Edge 集群
NCP00071	IP 池更新失败
NCP00072	调度程序失败
NCP00073	NAT 规则删除失败
NCP00074	逻辑路由器端口获取失败
NCP00075	NSX 配置验证失败

错误代码	说明
NCP00076	SNAT 规则更新失败
NCP00077	SNAT 规则重叠
NCP00078	负载均衡器端点添加失败
NCP00079	负载均衡器端点更新失败
NCP00080	负载均衡器规则池创建失败
NCP00081	找不到负载均衡器虚拟服务器
NCP00082	IP 集读取失败
NCP00083	SNAT 池获取失败
NCP00084	负载均衡器服务创建失败
NCP00085	负载均衡器服务更新失败
NCP00086	逻辑路由器端口更新失败
NCP00087	负载均衡器初始化失败
NCP00088	IP 池不唯一
NCP00089	第 7 层负载均衡器缓存同步错误
NCP00090	负载均衡器池不存在
NCP00091	负载均衡器规则缓存初始化错误
NCP00092	SNAT 进程失败
NCP00093	负载均衡器默认证书错误
NCP00094	负载均衡器端点删除失败
NCP00095	找不到项目
NCP00096	池访问被拒绝

错误代码	说明
NCP00097	无法获取负载均衡器服务
NCP00098	无法创建负载均衡器服务
NCP00099	负载均衡器池缓存同步错误

NSX 节点代理错误代码

错误代码	说明
NCP01001	找不到 OVS 上行链路
NCP01002	找不到主机 MAC
NCP01003	OVS 端口创建失败
NCP01004	无任何 pod 配置
NCP01005	Pod 配置失败
NCP01006	Pod 取消配置失败
NCP01007	找不到 CNI 套接字
NCP01008	CNI 连接失败
NCP01009	CNI 版本不匹配
NCP01010	CNI 消息接收失败
NCP01011	CNI 消息传输失败
NCP01012	Hyperbus 连接失败
NCP01013	Hyperbus 版本不匹配
NCP01014	Hyperbus 消息接收失败
NCP01015	Hyperbus 消息传输失败
NCP01016	GARP 发送失败
NCP01017	接口配置失败

nsx-kube-proxy 错误代码

错误代码	说明
NCP02001	代理网关端口无效
NCP02002	代理命令失败
NCP02003	代理验证失败

CLI 错误代码

错误代码	说明
NCP03001	CLI 启动失败
NCP03002	CLI 套接字创建失败
NCP03003	CLI 套接字异常
NCP03004	CLI 客户端请求无效
NCP03005	CLI 服务器传输失败
NCP03006	CLI 服务器接收失败
NCP03007	CLI 命令执行失败

Kubernetes 错误代码

错误代码	说明
NCP05001	Kubernetes 连接失败
NCP05002	Kubernetes 配置无效
NCP05003	Kubernetes 请求失败
NCP05004	找不到 Kubernetes 密钥
NCP05005	找不到 Kubernetes 类型
NCP05006	Kubernetes 监控程序异常
NCP05007	Kubernetes 资源长度无效
NCP05008	Kubernetes 资源类型无效
NCP05009	Kubernetes 资源句柄失败
NCP05010	Kubernetes 服务句柄失败
NCP05011	Kubernetes 端点句柄失败
NCP05012	Kubernetes Ingress 句柄失败
NCP05013	Kubernetes 网络策略句柄失败
NCP05014	Kubernetes 节点句柄失败
NCP05015	Kubernetes 命名空间句柄失败
NCP05016	Kubernetes pod 句柄失败
NCP05017	Kubernetes 密钥句柄失败
NCP05018	Kubernetes 默认后端失败

错误代码	说明
NCP05019	Kubernetes 匹配表达式不受支持
NCP05020	Kubernetes 状态更新失败
NCP05021	Kubernetes 注释更新失败
NCP05022	找不到 Kubernetes 命名空间缓存
NCP05023	找不到 Kubernetes 密钥
NCP05024	Kubernetes 默认后端正在使用中
NCP05025	Kubernetes LoadBalancer 服务句柄失败

Pivotal Cloud Foundry 错误代码

错误代码	说明
NCP06001	PCF BBS 连接失败
NCP06002	PCF CAPI 连接失败
NCP06006	找不到 PCF 缓存
NCP06007	PCF 域未知
NCP06020	PCF 策略服务器连接失败
NCP06021	PCF 策略处理失败
NCP06030	PCF 事件处理失败
NCP06031	PCF 意外事件类型
NCP06032	PCF 意外事件实例
NCP06033	PCF 任务删除失败
NCP06034	PCF 文件访问失败