Cisco IOS XR アプリケーション ホスティング コンフィギュレーション ガイド

初版：2015 年 12 月 23 日

シスコシステムズ合同会社
〒107-6227 東京都港区赤坂9-7-1 ミッドタウン・タワー
http://www.cisco.com/jp
お問い合わせ先：シスコ コンタクトセンター
0120-092-255 （フリーコール、携帯・PHS含む）
電話受付時間：平日 10:00～12:00、13:00～17:00
http://www.cisco.com/jp/go/contactcenter/

このマニュアルに記載されている仕様および製品に関する情報は、予告なしに変更されることがあります。このマニュアルに記載されている表現、情報、および推奨事項は、すべて正確であると考えていますが、明示的であれ黙示的であれ、一切の保証の責任を負わないものとします。このマニュアルに記載されている製品の使用は、すべてユーザ側の責任になります。

対象製品のソフトウェア ライセンスおよび限定保証は、製品に添付された『Information Packet』に記載されています。添付されていない場合には、代理店にご連絡ください。

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system.All rights reserved.Copyright © 1981, Regents of the University of California.

ここに記載されている他のいかなる保証にもよらず、各社のすべてのマニュアルおよびソフトウェアは、障害も含めて「現状のまま」として提供されます。シスコおよびこれら各社は、商品性の保証、特定目的への準拠の保証、および権利を侵害しないことに関する保証、あるいは取引過程、使用、取引慣行によって発生する保証をはじめとする、明示されたまたは黙示された一切の保証の責任を負わないものとします。

いかなる場合においても、シスコおよびその供給者は、このマニュアルの使用または使用できないことによって発生する利益の損失やデータの損傷をはじめとする、間接的、派生的、偶発的、あるいは特殊な損害について、あらゆる可能性がシスコまたはその供給者に知らされていても、それらに対する責任を一切負わないものとします。

このマニュアルで使用している IP アドレスおよび電話番号は、実際のアドレスおよび電話番号を示すものではありません。マニュアル内の例、コマンド出力、ネットワーク トポロジ図、およびその他の図は、説明のみを目的として使用されています。説明の中に実際のアドレスおよび電話番号が使用されていたとしても、それは意図的なものではなく、偶然の一致によるものです。

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries.To view a list of Cisco trademarks, go to this URL: http://www.cisco.com/go/trademarks.Third-party trademarks mentioned are the property of their respective owners.The use of the word partner does not imply a partnership relationship between Cisco and any other company.(1110R)

マニュアルの入手、Cisco Bug Search Tool（BST）の使用、サービス要求の送信、追加情報の収集の詳細については、『What's New in Cisco Product Documentation』を参照してください。

新しく作成された、または改訂されたシスコのテクニカル コンテンツをお手元に直接送信するには、『What's New in Cisco Product Documentation』RSS フィードをご購読ください。RSS フィードは無料のサービスです。

目 次

第 1 章

アプリケーション ホスティング向け Linux

ここでは、アプリケーションのホスティングと、Cisco IOS XR オペレーティング システムでのアプリケーションのホスティングに使用する Linux 環境について説明します。

アプリケーション ホスティングの必要性

過去10年間、既存のツールチェーンとのシームレスな統合による運用上の俊敏性と効率をサポートするネットワーク オペレーティング システムが求められてきました。サービス プロバイダーは、短期の製品サイクル、俊敏なワークフロー、およびモジュール型ソフトウェアを求めていました。これらすべてを効率的に自動化することが可能です。以前の 32 ビット QNX バージョンに代わる 64 ビットの Cisco IOS XR はこれらのすべての要件を満たします。これは、アプリケーションや設定管理ツール、業界標準のゼロ タッチ プロビジョニング メカニズムの統合を簡略にする環境を提供することで実現します。64 ビットの IOS XR はサービス プロバイダー向け DevOps 形式のワークフローに一致し、アプリケーションをホストするデバイスの設定と運用を自動化するために使用できるオープンな内部データ ストレージ システムを備えています。

仮想環境への移行は加速しているものの、再利用可能で、ポータブルで、スケーラブルなアプリケーションを構築する必要性が高まっています。アプリケーションのホスティングによって、管理者には独自のツールやユーティリティを利用するためのプラットフォームが与えられます。Cisco IOS XR 6.0 は Linux ツール チェーンを使用して構築されたサードパーティ製の市販アプリケーションをサポートしています。ユーザは、シスコが提供するソフトウェア開発キットと相互にコンパイルされたカスタムアプリケーションを実行できます。アプリケーションホスティングは、ネイティブとコンテナという 2 つの形態で提供されます。ネットワーク デバイスでホストされるアプリケーションは、さまざまな用途に利用できます。これは、既存のツールのチェーンによる自動化から、設定管理のモニタリング、統合に及びます。

アプリケーションをデバイス上でホストできるようにするには、次の要件を満たす必要があります。

- アプリケーションの構築に適切なビルド環境
- デバイス外のデバイスおよびネットワークと対話するためのメカニズム

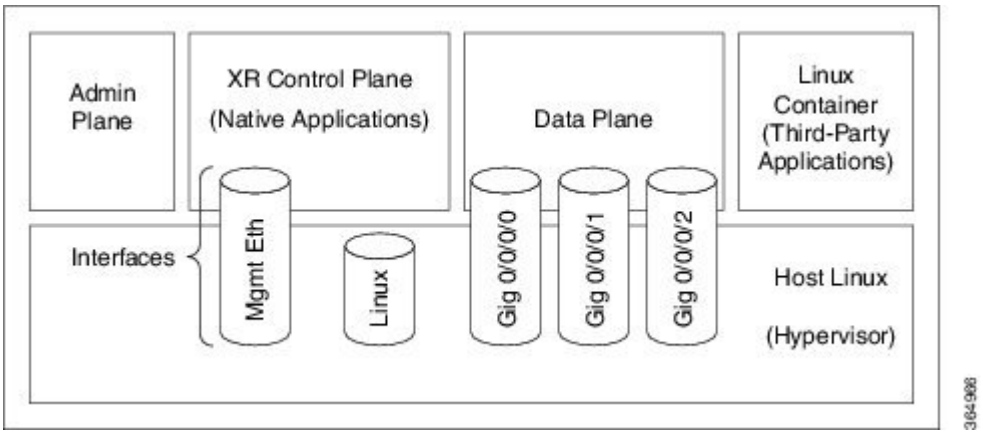
ネットワーク デバイスを、Chef や Puppet などの設定管理アプリケーションで管理すると、ネットワーク管理者は CLI のみに集中する作業から解放されます。アプリケーションが提供する抽象化により、アプリケーションがジョブを実行し、管理者は設計やその他の高レベルのタスクに集中できるようになりました。

アプリケーション ホスティングのアーキテクチャ

IOS XR は、ハイパーバイザを通じてアプリケーション ホスティング用の Linux コンテナを提供します。各コンテナによってユニークな機能が提供されます。64 ビットのホスト Linux（ハイパーバイザ）は Wind River Yocto のディストリビューションに基づいており、組み込み型システムで適切に動作します。ここでは、ホスト Linux 上で提供されるさまざまなコンテナを説明します。

次の図に、アプリケーション ホスティングのアーキテクチャを示します。

図 *1*：アプリケーション ホスティングのアーキテクチャ



- **管理プレーン**：管理プレーンは IOS XR の起動時に最初に起動される Linux コンテナです。管理プレーンは、IOS XR コントロール プレーン コンテナのライフ サイクルを管理します、
- **XR コントロール プレーン**：アプリケーションは、64 ビット IOS XR コントロール プレーンでネイティブにホストされます。コントロール プレーンを通じて IOS XR の Linux bash シェルにアクセスできます。
- **データ プレーン**：データ プレーンは、モジュラ ルータ シャーシ内のライン カードのすべての機能を代用し、提供します。
- **サードパーティ製コンテナ**：サードパーティ製アプリケーションをホストするための独自の Linux コンテナ（LXC）を作成し、提供されている LC インターフェイスを使用します。

Linux コンテナ以外に、ホストの Linux では複数のインターフェイスが提供されます。

IOS XR でのアプリケーションのホスティング向け Linux

Linux は、システム管理者、開発者、およびネットワーク エンジニアが過去 20 ～ 30 年にわたって作成し、テストし、導入してきたアプリケーションやツールのエコシステム全体をサポートします。Linuxは、アプリケーションの有無を問わず、安定性、セキュリティ、拡張性、低コストのライセンス、特定のインフラストラクチャのニーズに合わせたアプリケーションのカスタマイズを実現する柔軟性により、サーバのホスティングに適しています。

自動化と統合の簡易性に重点を置くDevOps形式のワークフローへの注目が高まる中、ネットワーク デバイスは進化し、自動化プロセスをより簡単にする標準的なツールやアプリケーションをサポートする必要があります。標準化された共有ツールのチェーンはスピード、効率、コラボレーションを強化できます。

IOS XR は Yocto ベースの Wind River 7 Linux ディストリビューションから開発されています。OS は RPM ベースとなっており、組み込み型システムに最適です。

IOS XR によって、ボックス上での 64 ビット Linux アプリケーションのホスティングが可能となり、次の利点が得られます。

- 設定管理アプリケーションとのシームレスな統合
- ファイル システムへの容易なアクセス
- 操作の簡易性

IOS XRの シンプルなアプリケーションのホスティングについては、アプリケーション ホスティングのタイプ，（11 ページ）を参照してください。

IOS XR の Linux シェルの概要

IOS XR の Linux アプリケーションをホストするには、XR の Linux シェルを理解している必要があります。

次のステップに従って、XR シェルを移動します。

1. Linux ボックスから SSH を使用して IOS XR コンソールにアクセスし、ログインします。

```
cisco@host:~$ ssh root@192.168.122.188
root@192.168.122.188's password:
RP/0/RP0/CPU0:ios#
```

   IOS XR プロンプトが表示されます。

2. IOS XR のイーサネット インターフェイスを表示します。

```
RP/0/0/CPU0:ios# show ipv4 interface brief
Wed Oct 28 18:45:56.168 IST

Interface                      IP-Address      Status                Protocol
Loopback0                      1.1.1.1/32      Up                    Up
GigabitEthernet0/0/0/0         10.1.1.1/24     Up                    Up
...


RP/0/RP0/CPU0:ios#show interfaces gigabitEthernet 0/0/0/0
```

```
Wed Oct 28 18:45:56.168 IST

GigabitEthernet0/0/0/0 is up, line protocol is up
Interface state transitions: 4
Hardware is GigabitEthernet, address is 5246.e8a3.3754 (bia
5246.e8a3.3754)
Internet address is 10.1.1.1/24
MTU 1514 bytes, BW 1000000 Kbit (Max: 1000000 Kbit)
reliability 255/255, txload 0/255, rxload 0/255
Encapsulation ARPA,
Duplex unknown, 1000Mb/s, link type is force-up
output flow control is off, input flow control is off
loopback not set,
Last link flapped 01:03:50
ARP type ARPA, ARP timeout 04:00:00
Last input 00:38:45, output 00:38:45
Last clearing of "show interface" counters never
5 minute input rate 0 bits/sec, 0 packets/sec
5 minute output rate 0 bits/sec, 0 packets/sec
12 packets input, 1260 bytes, 0 total input drops
0 drops for unrecognized upper-level protocol
Received 2 broadcast packets, 0 multicast packets
0 runts, 0 giants, 0 throttles, 0 parity
0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
12 packets output, 1224 bytes, 0 total output drops
Output 1 broadcast packets, 0 multicast packets
```

出力には、`GigabitEthernet0/0/0/0` インターフェイスの IP アドレスと MAC アドレスが表示されます。

3 **run** コマンドを入力し、IOS XR の Linux bash シェルを起動します。

また、bash プロンプトの表示時に IOS XR のバージョンも確認します。

```
RP/0/RP0/CPU0:ios# run
Wed Oct 28 18:45:56.168 IST

[xr-vm_node0_RP0_CPU0:~]$ uname -a
Linux xr-vm_node0_RP0_CPU0 3.10.19-WR7.0.0.2_standard #1 SMP Mon Jul 6
13:38:23 PDT 2015 x86_64 GNU/Linux
[xr-vm_node0_RP0_CPU0:~]$
```

（注） Linux bash セルを終了し、IOS XR コンソールを起動するには、**exit** コマンドを入力します。

```
[xr-vm_node0_RP0_CPU0:~]$exit
exit
RP/0/RP0/CPU0:ios#
```

4 **ifconfig** コマンドを実行してネットワーク インターフェイスを見つけます。

```
[xr-vm_node0_RP0_CPU0:~]$ifconfig
eth0      Link encap:Ethernet  HWaddr 52:46:12:7a:88:41
          inet6 addr: fe80::5046:12ff:fe7a:8841/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:8996  Metric:1
          RX packets:280 errors:0 dropped:0 overruns:0 frame:0
          TX packets:160 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:31235 (30.5 KiB)  TX bytes:20005 (19.5 KiB)

eth-vf0   Link encap:Ethernet  HWaddr 52:54:00:34:29:44
          inet addr:10.11.12.14  Bcast:10.11.12.255  Mask:255.255.255.0
          inet6 addr: fe80::5054:ff:fe34:2944/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:9000  Metric:1
          RX packets:19 errors:0 dropped:0 overruns:0 frame:0
          TX packets:13 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1566 (1.5 KiB)  TX bytes:1086 (1.0 KiB)
```

```
eth-vf1   Link encap:Ethernet  HWaddr 52:54:00:ee:f7:68
          inet6 addr: fe80::5054:ff:feee:f768/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:9000  Metric:1
          RX packets:326483 errors:0 dropped:3 overruns:0 frame:0
          TX packets:290174 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:24155455 (23.0 MiB)  TX bytes:215862857 (205.8 MiB)

eth-vf1.1794 Link encap:Ethernet  HWaddr 52:54:01:5c:55:8e
          inet6 addr: fe80::5054:1ff:fe5c:558e/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:8996  Metric:1
          RX packets:10 errors:0 dropped:0 overruns:0 frame:0
          TX packets:13 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:728 (728.0 B)  TX bytes:1234 (1.2 KiB)

eth-vf1.3073 Link encap:Ethernet  HWaddr e2:3a:dd:0a:8c:06
          inet addr:192.0.0.4  Bcast:192.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::e03a:ddff:fe0a:8c06/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:8996  Metric:1
          RX packets:317735 errors:0 dropped:3560 overruns:0 frame:0
          TX packets:257881 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:18856325 (17.9 MiB)  TX bytes:204552163 (195.0 MiB)

eth-vf1.3074 Link encap:Ethernet  HWaddr 4e:41:50:00:10:01
          inet addr:172.0.16.1  Bcast:172.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::4c41:50ff:fe00:1001/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:8996  Metric:1
          RX packets:8712 errors:0 dropped:0 overruns:0 frame:0
          TX packets:32267 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:723388 (706.4 KiB)  TX bytes:11308374 (10.7 MiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:1635360 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1635360 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:182532711 (174.0 MiB)  TX bytes:182532711 (174.0 MiB)

tap123    Link encap:Ethernet  HWaddr c6:13:74:4b:dc:e3
          inet6 addr: fe80::c413:74ff:fe4b:dce3/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:13 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:998 (998.0 B)
```

出力には、IOS XR が使用する内部インターフェイス（`eth0` ～ `eth-vf1.3074`）が表示されます。これらのインターフェイスは、XR ネットワークの名前空間（XRNNS）にあり、IOS XR の外部のネットワークとやり取りすることはありません。IOS XR の外部のネットワークとやり取りするインターフェイスは、サードパーティ製ネットワークの名前空間（TPNNS）にあります。TPNNS については、IOS XR のサードパーティ製ネットワークの名前空間, （6 ページ）を参照してください。

Linux ネットワークの名前空間の概要

一般的な Linux OS は、ネットワーク インターフェイスと OS で共有されるルーティング テーブル エントリ一式を提供します。ネットワークの名前空間の導入により、Linux は 独立して機能する複数のインスタンスのネットワーク インターフェイスとルーティング テーブルを提供します。

（注） ネットワークの名前空間のサポートは、Linux OS のディストリビューションによって異なります。アプリケーションのホスティングに使用するディストリビューションがネットワークの名前空間をサポートしていることを確認します。

ネットワークの名前空間に移動するには、次のコマンドを使用します。

```
ip netns exec <namespace_name>
```

IOS XR のサードパーティ製ネットワークの名前空間

IOS XR の Linux シェルは、サードパーティ製アプリケーションと内部 XR プロセス間に必要な隔離を実装すると同時に、XR インターフェイスへの必要なアクセスをアプリケーションに提供するサードパーティ製ネットワークの名前空間を提供します。

次に、IOS XR で TPNNS を表示する例を示します。

1. IOS XR の bash シェルで TPNNS に移動します。

```
[XR-vm_node0_RP0_CPU0:~]$ ip netns exec tpnns bash
```

2. TPNNS インターフェイスを表示します。

```
[XR-vm_node0_RP0_CPU0:~]$ ifconfig
Gi0_0_0_0 Link encap:Ethernet  HWaddr 52:46:04:87:19:3c
          inet addr:192.164.168.10  Mask:255.255.255.0
          inet6 addr: fe80::5046:4ff:fe87:193c/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

Mg0_RP0_CPU0_0 Link encap:Ethernet  HWaddr 52:46:12:7a:88:41
          inet addr:192.168.122.197  Mask:255.255.255.0
          inet6 addr: fe80::5046:12ff:fe7a:8841/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

fwd_ew    Link encap:Ethernet  HWaddr 00:00:00:00:00:0b
          inet6 addr: fe80::200:ff:fe00:b/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2 errors:0 dropped:1 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:140 (140.0 B)

fwdintf   Link encap:Ethernet  HWaddr 00:00:00:00:00:0a
          inet6 addr: fe80::200:ff:fe00:a/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1482  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2 errors:0 dropped:1 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:140 (140.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
```

```
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo:0      Link encap:Local Loopback
          inet addr:1.1.1.1  Mask:255.255.255.255
          UP LOOPBACK RUNNING  MTU:1500  Metric:1
```

出力に表示されるインターフェイスはLinux環境でのIOS XRインターフェイスの複製です（同じMACアドレスとIPアドレスを備えています）。

- `Gi0_0_0_0` はIOS XR GigabitEthernet 0/0/0/0インターフェイスです。
- `Mg0_RP0_CPU0_0` はXRでの管理操作に使用するIOS XR管理インターフェイスです。
- `fwd_ew` はサードパーティ製アプリケーションとIOS XR間の通信（水平方向）に使用するインターフェイスです。
- `fwdintf` はIOS XRの外部のサードパーティ製アプリケーションとネットワーク間の通信に使用するインターフェイスです。
- `lo:0` は `fwdintf` インターフェイスを通じたサードパーティ製アプリケーションと外部ネットワーク間の通信に使用するIOS XR loopback0インターフェイスです。loopback0インターフェイスは、XRの外部の通信に使用できるように設定する必要があります。また、IOS XR の外部との通信，（21 ページ）の項で説明したように、アプリケーションは外部通信用のGigEインターフェイスも設定できます。

（**no shut** コマンドで）イネーブルになっているすべてのインターフェイスがIOSXRのTPNNSに追加されます。

3 （任意）`fwd_ew` インターフェイスと `fwdintf` インターフェイスで使用されるIPアドレスを表示します。

```
[xr-vm_node0_RP0_CPU0:~]$ ip route
default dev fwdintf  scope link  src 1.1.1.1
8.8.8.8 dev fwd_ew  scope link
192.168.122.0/24 dev Mg0_RP0_CPU0_0  proto kernel  scope link  src 192.168.122.213
```

IOS XR でのサードパーティ製ネットワーク名前空間に移動する代替方法

IOS XR へのログイン時に **ip netns exec tpnns bash** コマンドを入力せずにTPNNSに直接移動するには、次に示すステップで説明する `sshd_tpnns` サービスを使用します。この手順には、サービスにアクセスするための非ルートユーザの作成が含まれています（ルートユーザはこのサービスにアクセスできません）。

（注） IOS XR で、インターフェイスをバインドするサービスを開始する前に、インターフェイスが設定され、起動され、動作可能であることを確認します。

インターフェイスが設定された後にのみサービスを開始するようにするには、サービス スクリプトに次の関数を含めます。

```
. /etc/init.d/tpnns-functions
tpnss_wait_until_ready
```

tpnss_wait_until_ready 関数を追加することによって、サービス スクリプトが 1 つ以上のインターフェイスが設定されるのを待ってから、サービスを開始するようになります。

1. （任意）リロード時に TPNNS サービスを自動的に開始するようにするには、`sshd_tpnns` サービスを追加し、そのサービスの存在を確認します。

```
bash-4.3# chkconfig --add sshd_tpnns
bash-4.3# chkconfig --list sshd_tpnns
sshd_tpnns      0:off   1:off   2:off   3:on    4:on    5:on    6:off
bash-4.3#
```

2. `sshd_tpnns` サービスを開始します。

```
bash-4.3# service sshd_tpnns start
Generating SSH1 RSA host key: [  OK  ]
Generating SSH2 RSA host key: [  OK  ]
Generating SSH2 DSA host key: [  OK  ]
  generating ssh ECDSA key...
Starting sshd: [  OK  ]

bash-4.3# service sshd_tpnns status
sshd (pid  6224) is running...
```

3. ステップ 1 で作成した非ルート ユーザとして `sshd_tpnns` セッションにログインします。

```
host@fe-ucs36:~$ ssh devops@192.168.122.222 -p 57722
devops@192.168.122.222's password:
Last login: Tue Sep  8 20:14:11 2015 from 192.168.122.1
XR-vm_node0_RP0_CPU0:~$
```

4. インターフェイスを表示して、TPNNS に移動していることを確認します。

```
[XR-vm_node0_RP0_CPU0:~]$ ifconfig
Gi0_0_0_0 Link encap:Ethernet  HWaddr 52:46:04:87:19:3c
          inet addr:192.164.168.10  Mask:255.255.255.0
          inet6 addr: fe80::5046:4ff:fe87:193c/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

Mg0_RP0_CPU0_0 Link encap:Ethernet  HWaddr 52:46:12:7a:88:41
          inet addr:192.168.122.197  Mask:255.255.255.0
          inet6 addr: fe80::5046:12ff:fe7a:8841/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

fwd_ew    Link encap:Ethernet  HWaddr 00:00:00:00:00:0b
          inet6 addr: fe80::200:ff:fe00:b/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2 errors:0 dropped:1 overruns:0 carrier:0
```

```
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:140 (140.0 B)

fwdintf   Link encap:Ethernet  HWaddr 00:00:00:00:00:0a
          inet6 addr: fe80::200:ff:fe00:a/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1482  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2 errors:0 dropped:1 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:140 (140.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo:0      Link encap:Local Loopback
          inet addr:1.1.1.1  Mask:255.255.255.255
          UP LOOPBACK RUNNING  MTU:1500  Metric:1
```

第 2 章

IOS XR でのシンプルなアプリケーションのホスティング

ここでは、さまざまな種類のアプリケーションのホスティングについて説明し、iPerfなどのシンプルなアプリケーションを IOS XR でネイティブに、またはサードパーティ製コンテナ内でホストする方法を示します。

アプリケーション ホスティングのタイプ

IOS XR でのアプリケーションのホスティングは、2 つの形態で提供されます。

- **ネイティブ**：IOS XR が提供するコンテナ内でアプリケーションをホストできます。アプリケーションは、パッケージ マネージャとして RPM を使用するシスコ固有の Linux ディストリビューション（River Linux 7）で構築する必要があります。アプリケーションは、IOS XR のルート ファイル システムにあるライブラリを使用します。Chef や Puppet などの設定管理ツールを使用して、アプリケーションのインストールを自動化できます。
- **コンテナ**：IOS XR で独自のコンテナを作成し、そのコンテナ内でアプリケーションをホストします。アプリケーションは、Linux ディストリビューションを使用して開発できます。これは、IOS XR のルート ファイル システムが提供するものとは異なるシステム ライブラリを使用するアプリケーションに適しています。

アプリケーション ホスティングのタイプの選択

要件と次の基準に応じて、アプリケーション ホスティングのタイプを選択できます。

- **リソース**：ホストされているアプリケーションが消費するリソースの量を制御する必要がある場合は、制約事項を設定できるコンテナ モデルを選択する必要があります。ネイティブ

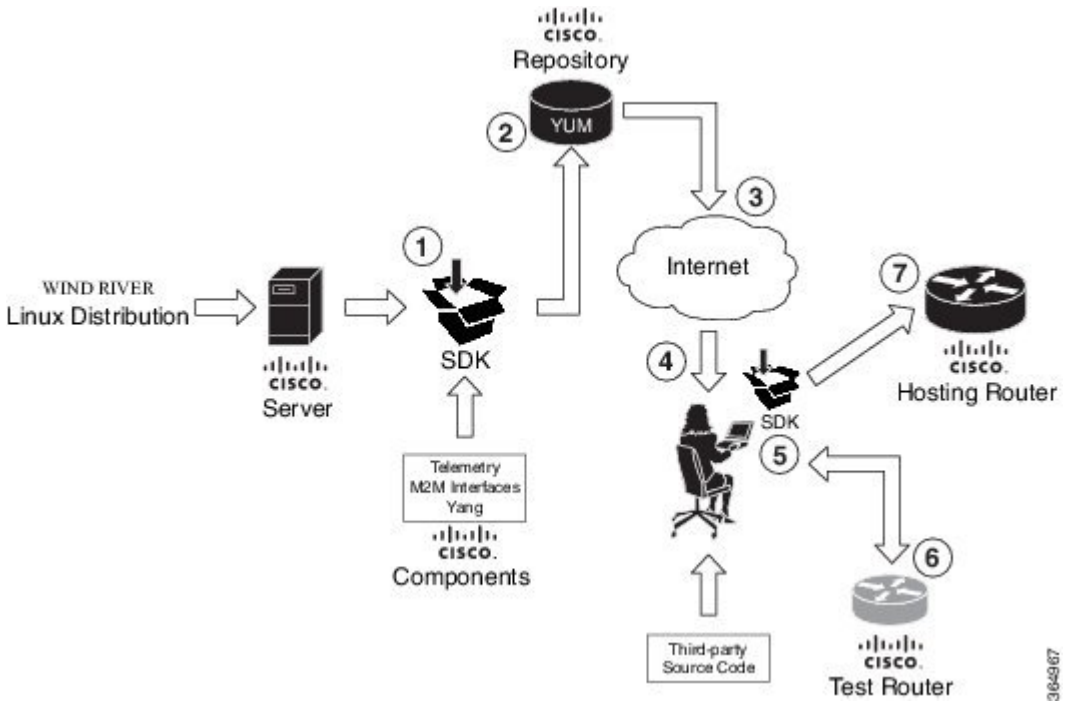
モデルでは、内部IOS XRプロセスと共有する割り当て済みのリソースを使用するアプリケーションのみを展開できます。

- **環境の選択**：ネイティブにホストされるアプリケーションは、IOS XRが提供するWind River 7 Linuxディストリビューションで構築する必要があります。アプリケーションの構築に使用するLinuxディストリビューションの選択肢を決定したら、コンテナ モデルを選択する必要があります。コンテナ モデルを使用してアプリケーションをホストすると、導入前にパッケージ化できます。

ネイティブ アプリケーションのホスティング：概要とワークフロー

ここでは、SDKシェルスクリプトを使用したネイティブアプリケーションのホスティングのワークフローを説明します。

図 *2*：*SDK* シェルス クリプトを使用したネイティブ アプリケーションのホスティング



1 SDKはYoctoプロジェクトのワークスペースを使用するWind River Linux 7（WRL7）ディストリビューションで構築されています。

IOS XRは、ベースの1つとしてYoctoプロジェクトからオープンな組み込み型カーネル インフラストラクチャを使用します。したがって、IOS XRには、すべてのシステム ライブラリと、アプリケーションをサポートするための基本的なツール チェーンが含まれています。その結果、IOS XRでネイティブにホストされるアプリケーションはWind Riverディストリビューショ

ン用に再構築する必要があります。また、テレメトリや YANG データ モデルなどの IOS XR の他の機能でそれらのアプリケーションをカスタマイズできます。

2 SDK はコンパイルされて、シスコの（YUM）リポジトリに保存されます。

（注） ネイティブ アプリケーションのホスティング環境は QEMU ハイパーバイザを使用するか、または SDK シェル スクリプトを実行するかのいずれかで構築されます。

Yellowdog Updated, Modified（YUM）は、Linux 用のオープン ソースのコマンドライン パッケージ管理ユーティリティであり、組み込み型のディストリビューション ソース ファイルの作成や Red Hat パケット マネージャ（RPM）などの適切な形式へのコンパイルが可能です。YUM では、自動のパケット更新が許可されます。YUM を使用することで、入手可能なパッケージをインストール、削除、更新、および表示することができます。

3 シスコのリポジトリは、インターネット上でのアプリケーション開発者によるアクセスを可能にします。

4 開発者は、SDK をダウンロードし、インストール用の SDK シェルス クリプトを実行します。詳細については、「SDK シェル スクリプトを使用した相互ビルド環境の作成, （26 ページ）」を参照してください。

5 開発者は IOS XR でホストされるアプリケーションを再構築します。詳細については、「ネイティブ RPM の構築, （27 ページ）」を参照してください。

6 開発者はテスト ルータで再構築されたアプリケーションをホストします。

7 開発者は、IOS XR を実行するホスティング ルータで再構築されたアプリケーションをホストします。

ネイティブ アプリケーションをホスティングするビルド環境の準備については、ビルド環境のセットアップ, （25 ページ）を参照してください。

ネイティブ アプリケーションとしての iPerf の実行

ネイティブ アプリケーションのホスティングの例として、この項で説明するように、IOS XR に iPerf クライアントをネイティブにインストールして、別のルータにネイティブにインストールされた iPerf サーバとの接続を確認することができます。

トポロジ

次の図に、この例で使用されるトポロジを説明します。

図 *3*：ネイティブ アプリケーションとしての *iPerf*

iPerf サーバはルータ A に、iPerf クライアントはルータ B にインストールされています。どちらのインストールも IOS RX でネイティブに実行されます。iPerf クライアントは、IOS XR が提供するインターフェイスを通じて iPerf サーバと通信します。

前提条件

2 つのルータがトポロジに示すように設定されていることを確認します。

設定手順

ネイティブ アプリケーションとして iPerf を実行するには、次の手順を実行します。

1. ルータ A にログインし、XRNNS に入ります。

```
RP/0/RP0/CPU0:ios# run
[xr-vm_node0_RP0_CPU0:~]$
```

2. ルータ A の RPM として iPerf サーバをインストールします。

```
[xr-vm_node0_RP0_CPU0:~]$yum install
https://s3.amazonaws.com/alpha-builds/iperf-2.0.5.ios_xr6.x86_64.rpm
```

3. ステップ 1 と 2 を実行し、iPerf クライアントをルータ B にインストールします。

4. ルータ A の iPerf サーバのインストールを確認します。

```
[xr-vm_node0_RP0_CPU0:~]$ iperf -v


iperf version 2.0.5 (08 Jul 2010) pthreads
```

   同様に、ルータ B の iPerf クライアントのインストールを確認します。

5. ルータ A の Loopback0 インターフェイスを iPerf サーバにバインドし、iPerf サーバ インスタンスを起動します。

   次の例では、1.1.1.1 はルータ A の Loopback0 インターフェイスの割り当てられたアドレス、57730 は通信に使用されるポート番号です。

```
[xr-vm_node0_RP0_CPU0:~]$ iperf -s -B 1.1.1.1 -p 57730
Server listening on TCP port 57730
Binding to local address 1.1.1.1
TCP window size: 85.3 KByte (default)
```

6 iPerf サーバに使用されているものと同じポート番号とルータ A の管理 IP アドレスを指定して、ルータ B の iPerf クライアント インスタンスを起動します。

次の例では、192.168.122.213 はルータ A の管理 IP アドレス、57730 は iPerf サーバへのアクセスに使用されるポート番号です。

```
[xr-vm_node0_RP0_CPU0:~]$ iperf -c 192.168.122.213 -p 57730
------------------------------------------------------------
Client connecting to 192.168.122.213, TCP port 57730
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 192.168.122.1 port 46974 connected with 192.168.122.213 port 57730
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec   146 MBytes   122 Mbits/sec
```

TCP ではなく UDP を使用して iPerf サーバと通信するには、次のコマンドを使用します。

```
[xr-vm_node0_RP0_CPU0:~]$ iperf -c 192.168.122.213 -p 57730 -u
------------------------------------------------------------
Client connecting to 192.168.122.213, UDP port 57730
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 192.168.122.1 port 41466 connected with 192.168.122.213 port 57730
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[  3] Sent 893 datagrams
[  3] Server Report:
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   0.233 ms    0/  893 (0%)
[root@hostB ~]#
```

7 ルータ B の iPerf クライアントから iPerf サーバに ping を実行します。

```
[xr-vm_node0_RP0_CPU0:~]$ /bin/ping 192.164.168.10
PING 192.164.168.10 (192.164.168.10) 56(84) bytes of data.
64 bytes from 192.164.168.10: icmp_seq=1 ttl=255 time=13.0 ms
64 bytes from 192.164.168.10: icmp_seq=2 ttl=255 time=2.14 ms
64 bytes from 192.164.168.10: icmp_seq=3 ttl=255 time=2.21 ms
```

ルータ B でホストされている iPerf クライアントはルータ A でホストされている iPerf サーバにアクセスします。

iPerf 用のネイティブ アプリケーションは正常にホスティングされています。

System V Init スクリプトを使用した iPerf の操作

iPerf のサーバ サービスまたはクライアント サービスは、IOS XR で自動的に起動、停止、または再起動できます。これを実現するには、次の例に示すように、System V（SysV）スクリプトを作成し、追加する必要があります。

1 iPerf サーバまたはクライアントを起動、停止、またはリロードする SysV スクリプトを作成します。

次の例では、iPerf サーバ用のスクリプトを作成します。また、iPerf クライアント用のスクリプトを作成するには、iPerf の設定手順のステップ 5 で説明したコマンドを使用します。

```
#!/bin/bash
#
# description: iPerf server
#
# Get function from functions library
. /etc/init.d/functions

# Start the service iperf
start() {
```

```
        iperf -s -B 1.1.1.1 -p 57730 & "Starting the iPerf Server: "
        /path/to/iperf &
        ### Create the lock file ###
        touch /var/lock/subsys/iperf
        success $"iPerf server startup"
        echo
}

# Restart the service iperf
stop() {
        iperf -s -B 1.1.1.1 -p 57730 & "Stopping the iPerf Server: "
        killproc iperf
        ### Delete the lock file ###
        rm -f /var/lock/subsys/iperf
        echo
}
### main logic ###
case "$1" in
  start)
        start
        ;;
  stop)
        stop
        ;;
  status)
        status iperf
        ;;
  restart|reload|condrestart)
        stop
        start
        ;;
  *)
        echo $"Usage: $0 {start|stop|restart|reload|status}"
        exit 1
esac
exit 0
```

2 iPerf サーバまたはクライアントをホストする IOS XR にスクリプトを追加します。

```
bash-4.3# chkconfig --add iperf
```

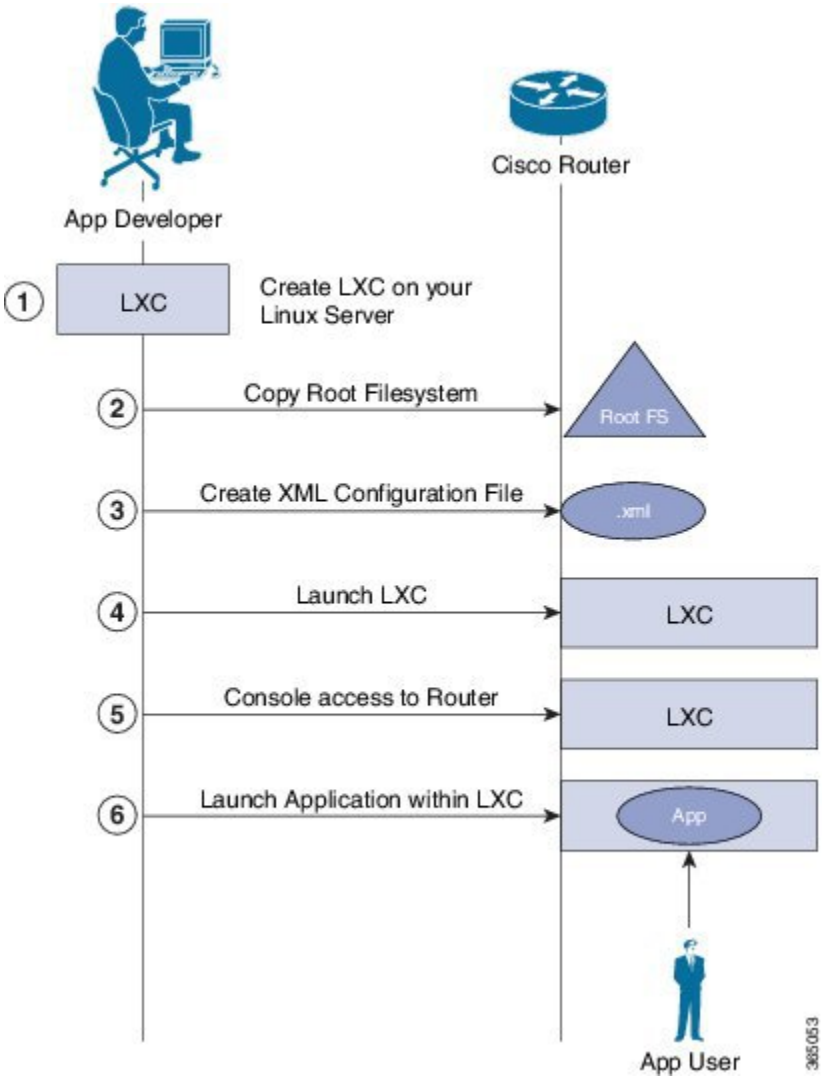
コンテナアプリケーションのホスティング：概要とワークフロー

ここでは、コンテナ アプリケーションのホスティングの概念を示し、そのワークフローについて説明します。

コンテナ アプリケーションのホスティングにより、アプリケーションが IOS XR の Linux コンテナ内の独自の環境とプロセス空間（名前空間）でホストできるようになります。アプリケーション開発者は、アプリケーション開発環境を完全に制御することができ、選択した Linux ディストリビューションを使用できます。アプリケーションは、IOS XR コントロール プレーン プロセスから隔離されていますが、XR GigE インターフェイスを通じて XR 外のネットワークに接続できます。また、アプリケーションは、IOS XR のローカル ファイル システムにも簡単にアクセスできます。

次の図に、アプリケーションをホストするための Linux コンテナを作成するワークフローを示します。完全な設定手順については、コンテナ アプリケーションとしての iPerf の実行，（18 ページ）を参照してください。

図 4：コンテナ アプリケーションのホスティングのワークフロー



コンテナ アプリケーションのホスティングには 2 つのコンポーネントがあります。

- **Linux サーバ**：アプリケーションを開発し、Linux コンテナ（LXC）を起動状態にし、コンテナ環境を準備するために使用するサーバです。
- **ルータ**：実行するアプリケーションでコンテナをホストするために使用される 64 ビットの IOS XR を実行するルータです。

1 Linux サーバ上で、LXC を起動し、次の手順を実行します。

a コンテナ環境と必要なライブラリを準備します。

b LXC をシャットダウンします。

2 IOS XR を実行するルータに接続し、ルート ファイル システムをコピーします。

3 コンテナの設定ファイルを .xml 形式で作成します。このファイルは、コンテナ名、デフォルトの名前空間などのコンテナの属性を指定します。

（注） サードパーティ製のネットワークの名前空間（TPNNS）を指定すると、デフォルトで LXC が TPNNS で起動します。

4 ルータの LXC を起動します。

5 IOS XR コンソール アクセスによりルータの LXC にログインします。

6 アプリケーションを手動で起動するか、または LXC の起動時にアプリケーションが自動的に起動するように設定します。

Linux ボックスなどのコンテナを使用してユーザ用のアプリケーションをインストールしたり、ホスティングしたりできます。

コンテナ アプリケーションとしての iPerf の実行

コンテナアプリケーションのホスティングの例として、この項で説明するように、IOS XR の LXC 内に iPerf クライアントをインストールして、別のルータの LXC 内に インストールされた iPerf サーバとの接続を確認することができます。

トポロジ

次の図で、この例で使用されるトポロジを説明します。

図 *5*：コンテナ アプリケーションとしての *iPerf*

iPerf サーバはルータ A に、iPerf クライアントはルータ B にインストールされています。どちらのインストールも 64 ビット IOS XR のコンテナ内で実行されます。iPerf クライアントは、IOS XR が提供するインターフェイスを通じて iPerf サーバと通信します。

前提条件

2 つのルータがトポロジに示すように設定されていることを確認します。

設定手順

アプリケーション コンテナとして iPerf を実行するには、次の手順を実行します。

1. ルータ A にログインし、XRNNS に入ります。

```
RP/0/RP0/CPU0:ios# run
[xr-vm_node0_RP0_CPU0:~]$
```

2. LXC を起動します。

```
[xr-vm_node0_RP0_CPU0:~]$virsh -c lxc+tcp://10.11.12.15:16509/ -e ^Q console demo1
```

3. 要求されたら、LXC にログインします。

```
Connected to domain demo
Escape character is ^Q
Kernel 3.14.23-WR7.0.0.2_standard on an x86_64

host login: Password:
```

4. ルータ A の LXC 内に iPerf サーバをインストールします。

```
[root@host ~]#apt-get install iperf
```

5. ステップ 1 ～ 4 を実行し、iPerf クライアントをルータ B にインストールします。

6. ルータ A の iPerf サーバのインストールを確認します。

```
[root@host ~]#iperf -v


iperf version 2.0.5 (08 Jul 2010) pthreads
```

   同様に、ルータ B の iPerf クライアントのインストールを確認します。

7. ルータ A の Loopback0 インターフェイスを iPerf サーバにバインドし、iPerf サーバ インスタンスを起動します。

   次の例では、1.1.1.1 はルータ A の Loopback0 インターフェイスの割り当てられたアドレス、57730 は通信に使用されるポート番号です。

```
[root@host ~]#iperf -s -B 1.1.1.1 -p 57730
Server listening on TCP port 57730
Binding to local address 1.1.1.1
TCP window size: 85.3 KByte (default)
```

8. iPerf サーバに使用されているものと同じポート番号とルータ A の管理 IP アドレスを指定して、ルータ B の iPerf クライアント インスタンスを起動します。

   次の例では、192.168.122.213 はルータ A の管理 IP アドレス、57730 は iPerf サーバへのアクセスに使用されるポート番号です。

```
[root@host ~]#iperf -c 192.168.122.213 -p 57730
------------------------------------------------------------
Client connecting to 192.168.122.213, TCP port 57730
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 192.168.122.1 port 46974 connected with 192.168.122.213 port 57730
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec   146 MBytes   122 Mbits/sec
```

TCP ではなく UDP を使用して iPerf サーバと通信するには、次のコマンドを使用します。

```
[root@host ~]#iperf -c 192.168.122.213 -p 57730 -u
------------------------------------------------------------
Client connecting to 192.168.122.213, UDP port 57730
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 192.168.122.1 port 41466 connected with 192.168.122.213 port 57730
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[  3] Sent 893 datagrams
[  3] Server Report:
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   0.233 ms    0/  893 (0%)
[root@hostB ~]#
```

9 ルータ B の iPerf クライアントから iPerf サーバに ping を実行します。

```
[root@host ~]#/bin/ping 192.164.168.10
PING 192.164.168.10 (192.164.168.10) 56(84) bytes of data.
64 bytes from 192.164.168.10: icmp_seq=1 ttl=255 time=13.0 ms
64 bytes from 192.164.168.10: icmp_seq=2 ttl=255 time=2.14 ms
64 bytes from 192.164.168.10: icmp_seq=3 ttl=255 time=2.21 ms
```

ルータ B でホストされている iPerf クライアントはルータ A でホストされている iPerf サーバにアクセスできます。

iPerf 用のコンテナ アプリケーションは正常にホスティングされています。コンテナ アプリケーションのホスティングの使用例については、IOS XR のコンテナ内でテレメトリ レシーバを実行する，（41 ページ）を参照してください。

第 3 章

ネットワーク スタックへのアクセス

Cisco IOS XR ソフトウェアは、通信を行うネットワーク スタックとして機能します。ここでは、IOS XR のアプリケーションが内部プロセスやサーバ、または外部デバイスとどのように通信を行うかについて説明します。

IOS XR の外部との通信

Cisco IOS XR の外部と通信するには、アプリケーションが `Loopback0` インターフェイスにマップされた `fwdintf` インターフェイス アドレス、または設定されたギガビット イーサネット インターフェイス アドレスを使用します。IOS XR のさまざまなインターフェイスについては、IOS XR のサードパーティ製ネットワークの名前空間，（6 ページ）を参照してください。

IOS XR の外部のそれぞれ対応するサーバとの IOS XR 通信に iPerf クライアントまたは Chef クライアントを配備するには、インターフェイス アドレスを XR の送信元アドレスとして設定する必要があります。リモート サーバは、IOS XR のそれぞれ対応するクライアントに到達するためにこのルート アドレスを設定する必要があります。

ここでは、外部通信の送信元アドレスとしてギガビット イーサネット インターフェイス アドレスを設定する例を示します。

外部通信用のギガビット イーサネット インターフェイスの使用法

外部通信用の IOS XR に GigE インターフェイスを設定するには、次の手順を実行します。

1 GigE インターフェイスを設定します。

```
RP/0/RP0/CPU0:ios(config)#interface GigabitEthernet 0/0/0/1
RP/0/RP0/CPU0:ios(config-if)#ipv4 address 192.57.43.10 255.255.255.0
RP/0/RP0/CPU0:ios(config-if)#no shut
RP/0/RP0/CPU0:ios(config-if)#commit
Fri Oct 30 07:51:14.785 UTC
RP/0/RP0/CPU0:ios(config-if)#exit
RP/0/RP0/CPU0:ios(config)#exit
```

2 設定されたインターフェイスが IOS XR で起動しており、動作可能かどうかを確認します。

```
RP/0/RP0/CPU0:ios#show ipv4 interface brief
Fri Oct 30 07:51:48.996 UTC

Interface                      IP-Address      Status                Protocol
Loopback0                      1.1.1.1         Up                    Up
Loopback1                      8.8.8.8         Up                    Up
GigabitEthernet0/0/0/0         192.164.168.10  Up                    Up
GigabitEthernet0/0/0/1         192.57.43.10    Up                    Up
GigabitEthernet0/0/0/2         unassigned      Shutdown              Down
MgmtEth0/RP0/CPU0/0            192.168.122.197 Up                    Up
RP/0/RP0/CPU0:ios#
```

3 Linux bash シェルを入力し、TPNNS に設定されているインターフェイスを確認します。

```
RP/0/RP0/CPU0:ios# run
[xr-vm_node0_RP0_CPU0:~]$ip netns exec tpnns bash
[xr-vm_node0_RP0_CPU0:~]$ifconfig
Gi0_0_0_0 Link encap:Ethernet  HWaddr 52:46:04:87:19:3c
          inet addr:192.164.168.10  Mask:255.255.255.0
          inet6 addr: fe80::5046:4ff:fe87:193c/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

Gi0_0_0_1 Link encap:Ethernet  HWaddr 52:46:2e:49:f6:ff
          inet addr:192.57.43.10  Mask:255.255.255.0
          inet6 addr: fe80::5046:2eff:fe49:f6ff/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

Mg0_RP0_CPU0_0 Link encap:Ethernet  HWaddr 52:46:12:7a:88:41
          inet addr:192.168.122.197  Mask:255.255.255.0
          inet6 addr: fe80::5046:12ff:fe7a:8841/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:3 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:294 (294.0 B)  TX bytes:504 (504.0 B)

fwd_ew    Link encap:Ethernet  HWaddr 00:00:00:00:00:0b
          inet6 addr: fe80::200:ff:fe00:b/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:4 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:1 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:392 (392.0 B)  TX bytes:532 (532.0 B)

fwdintf   Link encap:Ethernet  HWaddr 00:00:00:00:00:0a
          inet6 addr: fe80::200:ff:fe00:a/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1482  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2 errors:0 dropped:1 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:140 (140.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:1500  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:672 (672.0 B)  TX bytes:672 (672.0 B)
```

```
lo:0      Link encap:Local Loopback
          inet addr:1.1.1.1  Mask:255.255.255.255
          UP LOOPBACK RUNNING  MTU:1500  Metric:1
```

4 Linux bash シェルを終了し、外部通信の送信元アドレスとして GigE インターフェイスを設定します。

```
[xr-vm_node0_RP0_CPU0:~]$exit

RP/0/RP0/CPU0:ios#config
Fri Oct 30 08:55:17.992 UTC
RP/0/RP0/CPU0:ios(config)#tpa address-family ipv4 update-source gigabitEthernet 0/0/0/1
RP/0/RP0/CPU0:ios(config)#commit
Fri Oct 30 08:55:38.795 UTC
```

（注） デフォルトでは、`fwdintf` インターフェイスが外部通信用の `loopback0` インターフェイスにマップします。これは、`Loopback0` インターフェイスへのルーティング プロセスまたはルータ ID のバインドに似ています。`tpa address-family ipv4 update-source` コマンドを使用してギガビット イーサネット インターフェイスに `fwdintf` インターフェイスをバインドすると、インターフェイスがダウンした場合にネットワーク接続が影響を受けます。

5 Linux bash シェルを入力し、GigEインターフェイス アドレスが外部通信用に `fwdintf` インターフェイスによって使用されるかどうかを確認します。

```
RP/0/RP0/CPU0:ios#run
[xr-vm_node0_RP0_CPU0:~]$ip route
default dev fwdintf  scope link  src 192.57.43.10
8.8.8.8 dev fwd_ew  scope link
192.168.122.0/24 dev Mg0_RP0_CPU0_0  proto kernel  scope link  src 192.168.122.197
[xr-vm_node0_RP0_CPU0:~]$
```

IOS XR で外部通信が正常にイネーブルになります。

サードパーティ製アプリケーションの水平方向通信

IOS XR での水平方向通信は、コンテナでホストされているアプリケーションがネイティブの XR アプリケーション（XR コントロール プレーン上でホスティング）と対話するためのメカニズムです。

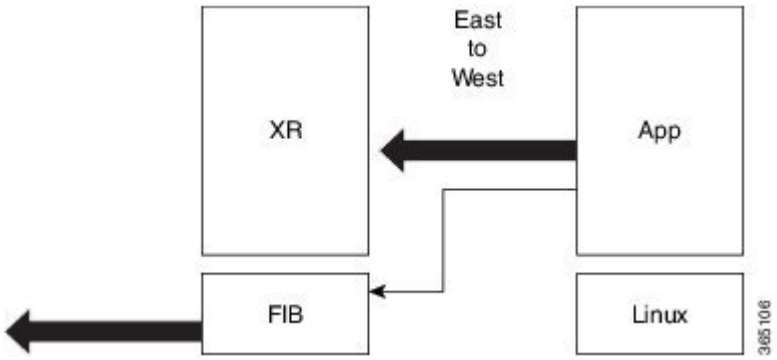
次の図に、IOS XR でホストされているサードパーティ製アプリケーションが XR コントロール プレーンとどのように対話するかを示します。

アプリケーションが IOS XR Forwarding Information Base（FIB）にデータを送信します。アプリケーションは、IOS XR の東側でホストされ、XR コントロール プレーンは西側の地域にあります。したがって、サードパーティ製アプリケーションと XR コントロール プレーン間の通信の形式は水平方向（E-W）通信と呼ばれます。

Chef Client や Puppet Agent などのサードパーティ製アプリケーションは、この通信モードを使用して、IOS XR 上でコンテナ、パケット、およびアプリケーションを設定し、管理します。今後、

このサポートがIOS XRに拡張されて、ようなサードパーティ製アプリケーションによって設定、制御される可能性があります。

図 *6*：*IOS XR* での水平方向の通信



IOS XR と通信するサードパーティ製アプリケーションでは、Loopback1 インターフェイスを設定する必要があります。次の手順でこの設定方法を説明します。

1. IOS XR の Loopback1 インターフェイスを設定します。

```
RP/0/RP0/CPU0:ios(config)#interface Loopback1
RP/0/RP0/CPU0:ios(config-if)#ipv4 address 8.8.8.8/32
RP/0/RP0/CPU0:ios(config-if)#no shut
RP/0/RP0/CPU0:ios(config-if)#commit
RP/0/RP0/CPU0:ios(config-if)#exit
RP/0/RP0/CPU0:ios(config)#
```

2. Loopback1 インターフェイスの作成を確認します。

```
RP/0/RP0/CPU0:ios#show ipv4 interface brief
Thu Nov 12 10:01:00.874 UTC

Interface                      IP-Address      Status                Protocol
Loopback0                      1.1.1.1         Up                    Up
Loopback1                      8.8.8.8         Up                    Up
GigabitEthernet0/0/0/0         192.164.168.10  Up                    Up
GigabitEthernet0/0/0/1         192.57.43.10    Up                    Up
GigabitEthernet0/0/0/2         unassigned      Shutdown              Down
MgmtEth0/RP0/CPU0/0            192.168.122.197 Up                    Up
RP/0/RP0/CPU0:ios#
```

3. IOS XR の Linux bash シェルを入力します。

```
RP/0/RP0/CPU0:ios#run
Thu Nov 12 10:03:38.699 UTC

[xr-vm_node0_RP0_CPU0:~]$
```

4. サードパーティ製ネットワークの名前空間（TPNNS）に移動します。

```
[xr-vm_node0_RP0_CPU0:~]$ip netns exec tpnns bash
[xr-vm_node0_RP0_CPU0:~]$
```

5. Loopback1 インターフェイス アドレスが E-W インターフェイスにマップされているかどうかを確認します。

```
[xr-vm_node0_RP0_CPU0:~]$ip route
default dev fwdintf  scope link  src 192.57.43.10
8.8.8.8 dev fwd_ew  scope link
192.168.122.0/24 dev Mg0_RP0_CPU0_0  proto kernel  scope link  src 192.168.122.197
[xr-vm_node0_RP0_CPU0:~]$
```

第 4 章

ネイティブ アプリケーションをホスティングするための RPM の構築

ここでは、ネイティブ アプリケーションをホストするための RPM の構築方法について説明します。

ビルド環境のセットアップ

ここでは、ネイティブアプリケーションホスティング対応にするビルド環境を準備し、設定する方法を説明します。

QEMU ハイパーバイザを使用したネイティブ ビルド環境の作成

ここでは、ネイティブ Wind River Linux 7.0 のビルド環境を作成し、クイック エミュレータ（QEMU）ハイパーバイザを使用して環境 ISO を実行する方法について説明します。

前提条件

- ネイティブ WRL7 ISO が含まれているシスコのリポジトリにアクセスできることを確認します。
- .iso 拡張子付きのネイティブ ISO をダウンロードします。

設定手順

1 ネイティブ WRL7 ISO を起動し、ディスク イメージにインストールします。

```
qemu-system-x86_64 -m 16G -cdrom <path-to-the-downloaded-iso-file> -net nic -net user
-hda ./wrl7.img
 -cpu core2duo -show-cursor -usb -usbdevice wacom-tablet -vga vmware
```

2 インストールされているイメージでネイティブ ビルド環境を再起動します。

```
qemu-system-x86_64 -m 16G -net nic -net user -hda ./wrl7.img -cpu core2duo -show-cursor
 -usb -usbdevice wacom-tablet -vga vmware
```

ネイティブ ビルド環境は、サードパーティ製アプリケーションをホストする準備が整っています。ユーザはネイティブ QEMU VM の VGA コンソール ポートに接続されます。

または、ユーザは VM 内で実行している SSH サービスに接続できます。

SDK シェル スクリプトを使用した相互ビルド環境の作成

WRL7 相互 SDK シェル スクリプトを使用して、相互ビルド環境をネイティブ環境の代わりとして作成することができます。Ubuntu 14.04 ホスト コンピュータなどの汎用 Linux 環境でシェルスクリプトを実行することによって、SDK をインストールできます。

前提条件

インストールを進める前に、次の要件が満たされていることを確認します。

- シスコのリポジトリの SDK へのアクセス。
- 必要に応じて、SDK にカスタマイズを構築できる機能。

インストール手順

ネイティブ アプリケーションをホスティングするための SDK をインストールするには、次の手順を実行します。

1 シスコのリポジトリから SDK をダウンロードします。

```
wget https://devhub.cisco.com/artifactory/xr600/app-dev-sdk/x86_64/
wrlinux-7.0.0.2-glibc-x86_64-intel_x86_64-wrlinux-image-glibc-std-sdk.sh
```

2 シェル スクリプトを実行して SDK をインストールします。

```
john@sjc-ads-4587:john$
./wrlinux-7.0.0.2-glibc-x86_64-intel_x86_64-wrlinux-image-glibc-std-sdk.sh
```

3 SDK のインストール先のディレクトリを入力します。

十分なストレージ領域があるターゲット ディレクトリを選択します。

```
Enter target directory for SDK
(default: /opt/windriver/wrlinux/7.0-intel-x86-64):
/nobackup/john/sdk_extract
You are about to install the SDK to "/nobackup/john/sdk_extract". Proceed[Y/n]? Y
```

正常にインストールされると、メッセージが画面に表示されます。

```
Extracting SDK...done
Setting it up...done
SDK has been successfully set up and is ready to be used.
```

ネイティブ アプリケーションのホスティングを行う SDK が正常にインストールされます。

次の作業

環境変数を設定して、この項で説明するようにそれらを検証することができます。

1 次のコマンドを実行することによって SDK がインストールされ、環境変数が設定されているディレクトリに移動します。

- bash シェルを使用する場合は、**. ./env.sh** コマンドを実行します。
- 他のシェルについては、**source . ./env.sh** コマンドを実行します。

コマンドは、SDK インストール時に抽出された環境設定ファイルを実行します。

2 **env** コマンドを実行してすべての変数値を表示し、インストールされている環境変数を検証します。

3 **env | grep CC** コマンドを実行し、次の値が割り当てられているかどうかを確認して CC 環境変数を検証します。

```
CC=x86_64-wrs-linux-gcc  -m64
--sysroot=/nobackup/john/sdk_extract/sysroots/core2-64-wrs-linux
```

または、echo コマンドを使用できます。

```
echo $CC
x86_64-wrs-linux-gcc -m64
--sysroot=/opt/windriver/wrlinux/7.0-intel-x86-64/sysroots/core2-64-wrs-linux
```

4 SDK がインストールされている場合、PATH 環境変数がベース ディレクトリを指しているかどうかを確認します。

パスを確認するには、**env | grep PATH** コマンドを実行して、次のパスが表示されるかどうかをチェックします。

```
PATH=<sdk_extract>/sysroots/
x86_64-wrlinuxsdk-linux/usr/bin:
<sdk_extract>/sysroots/x86_64-wrlinuxsdk-linux/usr/bin/x86_64-wrs-linux
```

または、echo コマンドを使用できます。

```
echo $PATH
<sdk_extract>/sysroots/x86_64-wrlinuxsdk-linux/usr/bin:
<sdk_extract>/sysroots/x86_64-wrlinuxsdk-linux/usr/bin/x86_64-wrs-linux
```

5 アプリケーションのソース コードを含むディレクトリに移動し、アプリケーションの構築を開始します。

（注） SDK のルート ファイル システムからすべての *.la 一時ファイルを削除する必要があります。これを実行するには、次のコマンドを使用します。

```
bash# cd  <sdk_extract>/sysroots/
bash# find . -name \*.la | xargs rm -f
```

ネイティブ RPM の構築

ここでは、ネイティブ環境または相互ビルド環境を使用してアプリケーションを構築する手順を説明します。ネイティブ ビルド環境を使用することを推奨します。

ソース コードからアプリケーションを構築するには、2 つの方法があります。

1 つは、ソース・コードのアーカイブからアプリケーションを構築する方法です。これは、この項で説明します。もう 1 つは、ソース RPM から構築する方法ですが、これは推奨されません。

前提条件

先に進む前に、次の要件が満たされていることを確認します。

- アプリケーションのビルド環境がネイティブ ビルド環境または相互ビルド環境のいずれかを使用するようにセットアップされていることを確認する。
- README ファイルを読み、アプリケーションの構築に関する構築プロセスを理解する。

設定手順

アプリケーションを構築するには、次の手順を実行します。

1. アプリケーションのソース コードが含まれているディレクトリに移動します。
2. 次のコマンドを実行して、アプリケーションを抽出します（圧縮されている場合）。

```
bash-4.1$ tar xzvf tcpdump-4.7.4.tar.gz
```

3. ディレクトリをアプリケーション ディレクトリに変更します。

```
bash-4.1$ cd tcpdump-4.7.4
```

4. アプリケーションを構築し、実行可能ファイルを生成します。

```
tcpdump-4.7.4$ ./configure
tcpdump-4.7.4$ make
```

5. ディレクトリにある実行可能ファイルを確認します。

```
tcpdump-4.7.4$ ls -1 ./tcpdump
-rwxr-xr-x 1 john eng 3677288 Jun 15 23:10 ./tcpdump
```

実行ファイルは `tcpdump` としてリストされています。

実行可能ファイルは、IOS XR でアプリケーションをホストするためにパッケージ化される準備が整っています。

次の作業

IOS XR にインストールできるようにアプリケーションのバイナリをパッケージ化します。

推奨されるパッケージ化の形式は、IOS XR でホスト可能な RPM です。

RPM を構築するには、次の手順を実行します。

- .spec ファイルが必要です。
- **rpmbuild** コマンドを実行する必要があります。

次の手順を実行して、バイナリをパッケージ化します。

1. SPECS ディレクトリに .spec ファイルを作成します。

```
# %define __strip /bin/true

Name: tcpdump
```

```
Version: 4.7.4
Release: XR
Buildroot: %{_tmppath}/%{name}-%{version}-%{release}-root
License: Copyright (c) 2015 Cisco Systems Inc. All rights reserved.
Packager: mark
SOURCE0 : %{name}-%{version}.tar.gz
Group: 3'rd party applicaiton
Summary: Tcpdump cross compiled for WRL6

%description
This is a cross compiled version of tcpdump using IOS XR sdk for WRL7

%prep

%setup -q -n %{name}-%{version}

%build
  # This where sdk is being sourced
source /nobackup/mark/sdk_extract_18/tmp/env.sh
 ./configure
 make


 %install
rm -rf ${RPM_BUILD_ROOT}
 # make DESTDIR=$RPM_BUILD_ROOT install
mkdir -p ${RPM_BUILD_ROOT}%{_sbindir}
 install -m755 tcpdump ${RPM_BUILD_ROOT}%{_sbindir}

%files
 %defattr(-,root,root)
 %{_sbindir}/tcpdump

%pre

 %post

 %preun

 %postun

%clean
rm -rf $RPM_BUILD_ROOT
```

2 RPM を構築します。

```
mark@tenby:redhat$ cd /usr/src/redhat/SPECS/
mark@tenby:SPECS$ rpmbuild -ba tcpdump.spec
```

使用する RPM のビルドは、5.4.14 バージョンです。

3 バイナリが RPMS ディレクトリに構築されていることを確認します。

```
mark@tenby:x86_64$ pwd /usr/src/redhat/RPMS/x86_64

mark@tenby:x86_64$ ls

tcpdump-4.7.4-XR.x86_64.rpm
```

ネイティブアプリケーションをホストする準備が整いました。ネイティブアプリケーションのホスティングについては、ネイティブ アプリケーションとしての iPerf の実行, （13 ページ）を参照してください。

第 5 章

設定管理ツールを使用したアプリケーションのホスティング

設定管理ツールは、サーバやネットワーク デバイスのセットアップなどの手動作業を自動化するために使用されます。アプリケーションの配信の要件として、ネットワーク機器の絶え間ない変更や再設定が問題となっています。手動による再設定プロセスがエラーを引き起こし、それがネットワークを停止させる原因となる可能性があります。設定に常に更新が必要で、しかもネットワーク デバイスが複数ある場合は、設定管理ツールが役に立ちます。

Cisco IOS XR ソフトウェアは、次の設定管理ツールと適切に機能することができます。

- Chef
- Puppet

ここでは、IOS XR でのアプリケーションのホスティングに適した設定管理ツールである Chef および Puppet をインストールし、設定し、使用する方法について説明します。

- ネイティブなアプリケーション ホスティング向け Chef, 31 ページ
- ネイティブなアプリケーション ホスティング向け Puppet, 35 ページ

ネイティブなアプリケーション ホスティング向け Chef

Chef は IOS XR でさまざまなアプリケーションをインストールし、設定し、ネイティブに導入できるオープンソースの IT 自動化ツールです。

IOS XR でアプリケーションをネイティブに導入するために Chef を使用するには、次のコンポーネントが必要です。

- Cisco IOS XR 6.0 向けの Chef Client RPM Version 12.5 以降
- Chef Server Version 12.4 以降
- IOS XR の Wind River Linux 7 環境に対応しているアプリケーション

また、IOS XR にアプリケーションをネイティブに導入するには、3 つの Chef ビルトイン リソースも必要です。3 つのビルトイン Chef リソースは次のとおりです。

- パッケージ リソース
- ファイル リソース
- サービス リソース

ChefおよびChefのリソースの詳細については、次の表に示されているリンクにアクセスしてください。

表 *1*：*Chef* のリソース

トピック	リンク
Chef Software, Inc.	https://www.chef.io/
Chef の概要	https://docs.chef.io/chef_overview.html
パッケージ リソースの参考資料	https://docs.chef.io/resource_package.html
ファイル リソースの参考資料	https://docs.chef.io/resource_file.html
サービス リソースの参考資料	https://docs.chef.io/resource_service.html
Chef Server の参考資料	https://docs.chef.io/install_server.html
ネイティブ XR 環境向け Chef Client	Chef Client

次の各項では、ネイティブなアプリケーション ホスティングを行うためにChefのレシピをどのようにインストールし、設定し、作成するかについて説明します。ネイティブなアプリケーション ホスティングについては、ネイティブ アプリケーションのホスティング：概要とワークフロー, （12 ページ）を参照してください。

Chef クライアントのインストールと設定

ここでは、IOS XR に Chef Client をインストールする手順について説明します。

前提条件

インストールを進める前に、次の要件が満たされていることを確認します。

- ワークステーションが Chef リポジトリと Chef 開発キットを使用してセットアップされている。

- Chef Server Version 12.4 以降がインストールされており、Linux ボックスからアクセスできる。
- Chef Server の ID ファイルを使用できる。
- Linux 環境（/etc/resolv.conf）で設定された正しいサーバ名とドメイン名のエントリがある。
- ルータが有効な NTP サーバを使用している。

設定手順

IOS XR に Chef Client をインストールして設定するには、次の手順を実行します。

1. Linux ボックスから SSH を使用して IOS XR コンソールにアクセスし、ログインします。

```
cisco@host:~$ ssh root@192.168.122.188
root@192.168.122.188's password:
RP/0/RP0/CPU0:ios#
```

IOS XR プロンプトが表示されます。

2. **run** コマンドを入力し、IOS XR の Linux bash シェルを起動します。

```
RP/0/RP0/CPU0:ios# run
Wed Oct 28 18:45:56.168 IST
[xr-vm_node0_RP0_CPU0:~]$
```

3. サードパーティ製ネットワークの名前空間（TPNNS）に移動します。

次に示すように、インターフェイスを表示し、TPNNS を使用しているかどうかを確認します。

```
xr-vm_node0_RP0_CPU0:~]$ip netns exec tpnns bash
[xr-vm_node0_RP0_CPU0:~]$ifconfig
Gi0_0_0_0 Link encap:Ethernet  HWaddr 52:46:04:87:19:3c
          inet addr:192.164.168.10  Mask:255.255.255.0
          inet6 addr: fe80::5046:4ff:fe87:193c/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

Mg0_RP0_CPU0_0 Link encap:Ethernet  HWaddr 52:46:12:7a:88:41
          inet addr:192.168.122.197  Mask:255.255.255.0
          inet6 addr: fe80::5046:12ff:fe7a:8841/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

fwd_ew    Link encap:Ethernet  HWaddr 00:00:00:00:00:0b
          inet6 addr: fe80::200:ff:fe00:b/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2 errors:0 dropped:1 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:140 (140.0 B)

fwdintf   Link encap:Ethernet  HWaddr 00:00:00:00:00:0a
          inet6 addr: fe80::200:ff:fe00:a/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1482  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2 errors:0 dropped:1 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:140 (140.0 B)
```

```
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo:0      Link encap:Local Loopback
          inet addr:1.1.1.1  Mask:255.255.255.255
          UP LOOPBACK RUNNING  MTU:1500  Metric:1

[xr-vm_node0_RP0_CPU0:~]$
```

4 （任意）必要に応じて、プロキシ サーバ（`http_proxy`、`https_proxy`）を設定します。

```
http_proxy=http://proxy.youtube.com:8080
https_proxy=https://proxy.youtube.com:8080
```

5 Chef Client をインストールします。

```
[xr-vm_node0_RP0_CPU0:~]$yum install https://chef.io/chef/install.sh
```

Chef の **install.sh** スクリプトによって、インストール用の最新バージョンの Chef Client RPM が自動的に特定されます。

6 `validation.pem` ファイルを Chef サーバから `/etc/chef/validation.pem` にコピーします。

7 `/etc/chef/client.rb` で Chef Server の ID とクライアントの設定を使用して Chef Client の設定を編集します。

```
validation_client_name 'chef-validator'
chef_server_url 'https://my_chef_server.youtube.com/organizations/chef'
node_name 'n3k.youtube.com' # "This" client device.
cookbook_sync_threads 5 # necessary for small memory switches (4G or less)
interval 30 # client-run interval; remove for "never"
```

8 Chef Client を実行します。

```
[xr-vm_node0_RP0_CPU0:~]$chef-client
```

（注） クライアントを一度実行するには、**chef-client --once** コマンドを使用します。詳細については、https://docs.chef.io/chef_client.html にある Chef のマニュアルを参照してください。

Chef Client が IOS XR に正常にインストールされます。

レシピによる Chef クックブックの作成

Chef レシピとともにロードされた Chef クックブックを Linux ワークステーションに作成して、Chef サーバにコピーできます。IOS XR に Chef クライアントをインストールした後、Chef サーバからクックブックをレシピとともにダウンロードし、クライアント実行中に使用することができます。

前提条件

先に進む前に、次の要件が満たされていることを確認します。

- ネイティブ IOS XR 環境に対応するアプリケーション パッケージにアクセスできる。
- ターゲット アプリケーション パッケージがアクセス可能なリポジトリでホストされているか、ブート フラッシュにダウンロードされている。

設定手順

次の手順を使用して、`bootlogd` サービスを開始し IOS XR の iPerf にインストールする Chef レシピを作成します。

1. 対応する knife コマンドを使用して Linux ワークステーションにクックブックを作成します。

```
knife cookbook create cisco-network-chef-cookbook
```

2. iPerf をインストールする Chef レシピ ファイルを作成し、クックブックに追加します。

   Chef レシピを `cisco-network-chef-cookbook/recipes/` ディレクトリに作成する必要があります。Chef クライアントによって自動的にロードされるようにするには、Chef レシピの名前を `default.rb` にする必要があります。

```
#
# Recipe:: demo_default_providers
#
# Copyright (c) 2015 The Authors, All Rights Reserved.

package = 'iperf-2.0.5-r0.0.core2_64.rpm'
service = 'bootlogd'

remote_file "/#{package}" do
  source "http://10.105.247.73/wrl7_yum_repo/#{package}"
  action :create
end

yum_package "#{package}" do
  source "/#{package}"
  action :install
end

service "#{service}" do
  action :start
end
```

3. Linux ワークステーションから Chefサーバにアクセスし、クックブックをサーバにアップロードします。
4. IOS XR シェルにログインし、Chef クライアントを実行してクックブックをロードし、実行します。

```
[xr-vm_node0_RP0_CPU0:~]$chef-client
```

iperf RPM が IOS XR にインストールされます。

Chef クライアントの詳細については、https://docs.chef.io/chef_client.html を参照してください。

ネイティブなアプリケーションホスティング向けPuppet

Puppet は IOS XR でさまざまなアプリケーションをインストールし、設定し、ネイティブに導入できるオープンソースの IT 自動化ツールです。

IOS XR でアプリケーションをネイティブに導入するために Puppet を使用するには、次のコンポーネントが必要です。

- IOS XR 用に構築された Puppet RPM。
- Puppet Master Version 4.0 以降。
- IOS XR の Wind River Linux 7 環境に対応しているアプリケーション

また、IOS XR にアプリケーションをネイティブに導入するには、3 つの Puppet ビルトイン タイプも必要です。3 つのビルトイン Puppet タイプは次のとおりです。

- Package Type
- File Type
- Service Type

Puppet の詳細については、次の表に示されているリンクにアクセスしてください。

表 *2*：*Puppet* のリソース

トピック	Link
Puppet Labs	https://puppetlabs.com/
Puppet の概要	https://docs.puppetlabs.com/pe/latest/puppet_overview.html
パッケージ タイプの参考資料	https://docs.puppetlabs.com/references/latest/type.html#package
ファイル タイプの参考資料	https://docs.puppetlabs.com/references/latest/type.html#file
サービス タイプの参考資料	https://docs.puppetlabs.com/references/latest/type.html#service
Puppet Master の参考資料	Puppet Master
ネイティブ IOS XR 環境向けの Puppet Agent	Puppet Agent

次の各項では、ネイティブなアプリケーション ホスティングを行うために Puppet のマニフェストをどのようにインストールし、設定し、作成するかについて説明します。ネイティブなアプリケーション ホスティングについては、ネイティブアプリケーションのホスティング：概要とワークフロー，（12 ページ）を参照してください。

Puppet Agent のインストールと設定

ここでは、IOS XR で Puppet Agent をインストールし、設定する方法について説明します。

前提条件

インストールを進める前に、次の要件が満たされていることを確認します。

- Puppet Master Version 4.0 以降がインストールされており、ワークステーションからアクセスできる。
- Puppet Master の ID ファイルを使用できる。
- Linux 環境（`/etc/resolv.conf`）で設定された正しいサーバ名とドメイン名のエントリがある。
- ルータが有効な NTP サーバを使用している。

設定手順

IOS XR で Puppet Agent をインストールして設定するには、次の手順を実行します。

1. Linux ボックスから SSH を使用して IOS XR コンソールにアクセスし、ログインします。

```
cisco@host:~$ ssh root@192.168.122.188
root@192.168.122.188's password:
RP/0/RP0/CPU0:ios#
```

IOS XR プロンプトが表示されます。

2. **run** コマンドを入力し、IOS XR の Linux bash シェルを起動します。

```
RP/0/RP0/CPU0:ios# run
Wed Oct 28 18:45:56.168 IST
[xr-vm_node0_RP0_CPU0:~]$
```

3. サードパーティ製ネットワークの名前空間（TPNNS）に移動します。

に示すように、インターフェイスを表示して TPNNS を使用しているかどうかを確認します。

```
xr-vm_node0_RP0_CPU0:~]$ip netns exec tpnns bash
[xr-vm_node0_RP0_CPU0:~]$ifconfig
Gi0_0_0_0 Link encap:Ethernet  HWaddr 52:46:04:87:19:3c
          inet addr:192.164.168.10  Mask:255.255.255.0
          inet6 addr: fe80::5046:4ff:fe87:193c/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

Mg0_RP0_CPU0_0 Link encap:Ethernet  HWaddr 52:46:12:7a:88:41
          inet addr:192.168.122.197  Mask:255.255.255.0
          inet6 addr: fe80::5046:12ff:fe7a:8841/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

fwd_ew    Link encap:Ethernet  HWaddr 00:00:00:00:00:0b
          inet6 addr: fe80::200:ff:fe00:b/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2 errors:0 dropped:1 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:140 (140.0 B)

fwdintf   Link encap:Ethernet  HWaddr 00:00:00:00:00:0a
          inet6 addr: fe80::200:ff:fe00:a/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1482  Metric:1
```

```
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2 errors:0 dropped:1 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:140 (140.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo:0      Link encap:Local Loopback
          inet addr:1.1.1.1  Mask:255.255.255.255
          UP LOOPBACK RUNNING  MTU:1500  Metric:1
```

4 （任意）必要に応じて、プロキシ サーバ（`http_proxy`、`https_proxy`）を設定します。

```
http_proxy=http://proxy.youtube.com:8080
https_proxy=https://proxy.youtube.com:8080
```

5 Puppet Agent をインストールします。

```
[xr-vm_node0_RP0_CPU0:~]$ wget http://yum.puppetlabs.com/RPM-GPG-KEY-puppetlabs
[xr-vm_node0_RP0_CPU0:~]$ wget http://yum.puppetlabs.com/RPM-GPG-KEY-reductive
[xr-vm_node0_RP0_CPU0:~]$ rpm --import RPM-GPG-KEY-puppetlabs RPM-GPG-KEY-reductive
[xr-vm_node0_RP0_CPU0:~]$ yum install
http://yum.puppetlabs.com/puppetlabs-release-pc1-cisco-wrlinux-7.noarch.rpm
```

6 `/etc/puppetlabs/puppet/puppet.conf` で Puppet Master の ID および Agent の設定を使用して Puppet Agent のコンフィギュレーション ファイルを編集します。

Puppet Agent が IOS XR に正常にインストールされます。

Puppet マニュフェストの作成

ここでは、アプリケーションをホストするための Puppet マニフェストの作成方法について説明します。

前提条件

先に進む前に、次の要件が満たされていることを確認します。

- ネイティブ IOS XR 環境に対応するアプリケーション パッケージにアクセスできる。
- ターゲット アプリケーション パッケージがアクセス可能なリポジトリでホストされているか、ブート フラッシュにダウンロードされている。

設定手順

Puppet マニフェストを作成して `bootlogd` サービスを開始し、IOS XR に iPerf をインストールするには、次の手順を実行します。

1 Puppet Master に Puppet マニフェストを作成し、アプリケーションを導入します。

マニフェストは /etc/puppetlabs/code/environments/production/manifests/ ディレクトリに作成する必要があります。Puppet Masterによって自動的に起動するようにするには、マニフェストの名前を site.pp にする必要があります。

```
# Manifest to demo builtin providers
#

class ciscopuppet::demo_builtin_providers {

  $package = 'iperf'
  $service = 'bootlogd'

  yumrepo { 'wrl7-repo':
    ensure => present,
    name => 'wrl7-repo',
    baseurl => 'http://10.105.247.73/wrl7_yum_repo/',
    gpgcheck => 0,
    enabled => 1,
    proxy => '_none_',
  }

  package { $package:
    ensure => present,
    require => Yumrepo['wrl7-repo'],
  }

  service { $service:
    ensure => running,
  }

}

node 'default' {
  include ciscopuppet::demo_builtin_providers
}
```

2 Puppet Agent にアクセスしてトリガーし、Puppet Master に定義されているマニフェストに基づいてシステムを収束します。

iPerf RPM が Puppet Agent によって IOS XR にインストールされます。

第 6 章

使用例：コンテナ アプリケーションのホスティング

ここでは、IOS XR のコンテナ内でのテレメトリ レシーバの実行方法について説明します。コンテナアプリケーションのホスティングについては、コンテナアプリケーションのホスティング：概要とワークフロー，（16 ページ）を参照してください。

IOS XR のコンテナ内でテレメトリ レシーバを実行する

テレメトリを IOS XR で機能させるには、TCP ではなく、UDP を介して GPB（Google プロトコル バッファ）を使用する必要があります。

この手順は、次のステップから構成されています。

1. テレメトリ ポリシー ファイルの作成。
2. .proto ファイルの生成とコンパイル。
3. GPB エンコーダの設定。
4. サードパーティ製コンテナ（LXC）の起動。
5. テレメトリ レシーバの設定。

テレメトリ ポリシー ファイルの作成

テレメトリ ポリシー ファイルは、生成してからテレメトリ レシーバにプッシュするデータの種類を指定します。次に、テレメトリ用のポリシー ファイルの作成手順を示します。

1. データをストリーミングするスキーマ パスを決定します。

```
RP/0/RP0/CPU0:ios# schema-describe show interface
Wed Aug 26 02:24:40.556 PDT
RootOper.InfraStatistics.Interface(*).Latest.GenericCounters
```

2 これらのパスが含まれているポリシー ファイルを作成します。

```
{
"Name": "Test",
"Metadata": {
"Version": 25,
"Description": "This is a sample policy",
"Comment": "This is the first draft",
"Identifier": "<data that may be sent by the encoder to the mgmt stn"
},
"CollectionGroups": {
"FirstGroup": {
"Period": 30,
"Paths": [
"RootOper.InfraStatistics.Interface(*).Latest.GenericCounters"
   ]
  }
 }
}
```

3 XR Linux bash シェルを起動し、Secure Copy Protocol（SCP）を使用してポリシー ファイルを IOS XR にコピーします。.

```
RP/0/RP0/CPU0:ios# run
[XR-vm_node0_RP0_CPU0:~]$ ip netns exec tpnns bash
[XR-vm_node0_RP0_CPU0:~]$ scp Test.policy cisco@10.0.0.1:/telemetry/policies
cisco@10.0.0.1's password:
Test.policy
100% 779 0.8KB/s 00:00
Connection to 10.0.0.1 closed by remote host.
```

10.0.0.1 は、ポリシー ファイルのコピー元デバイスの IP アドレスです。

4 IOS XR プロンプトに移動し、ポリシー ファイルが正しくインストールされているかどうかを確認します。

```
RP/0/RP0/CPU0:ios# show telemetry policies brief
Wed Aug 26 02:24:40.556 PDT
Name |Active?| Version | Description
----------------------------|-------|---------|------------------------------
Test N 1 This is a sample policy
```

.proto ファイルの生成とコンパイル

作成したポリシー ファイルのパス内には、そのパスに関連付けられた `.proto` ファイルが必要です。.proto ファイルはデータのストリーミングに使用する GPB メッセージ形式を記述します。次に、テレメトリ レシーバ用に .proto ファイルを生成し、コンパイルするステップを示します。`.proto` ファイルは `.map` ファイルにコンパイルされます。コンパイルはサーバ上で実行されます。`telemetry_protoc` というコンパイラを http://gitlab.cisco.com/groups/telemetry で入手できます。このコンパイラは `.proto` ファイルを取得し、メッセージを符号化するためにルータ上で使用するコードを生成します。

1 .proto ファイルを生成します。

```
telemetry generate gpb-encoding path
"RootOper.InfraStatistics.Interface(*).Latest.GenericCounters" file
disk0:generic_counters.proto
```

.proto ファイルは、オンボックス ツールによって生成されます。ツールは命名パラメータを無視するため、任意です。

（注） ツールは引用符内のテキストを無視します。そのため、パスには引用符を含めないでください。

2 ボックス外で .proto ファイルをコンパイルします。

a シスコは Dev Hub でテレメトリ コンパイラを提供しています。Linux ボックスにディレクトリをコピーして、次に示すように実行することができます。

```
telemetry_protoc -f generic_counters.proto -o generic_counters.map
```

b .proto ファイルのコピーに Dev Hub からアクセスし、次のように Linux ボックスの標準コンパイラを実行します。

```
protoc python_out . -I=/
sw/packages/protoc/current/google/include/:.
generic_counters.proto ipv4_counters.proto
```

3 `/telemetry/gpb/maps` でマップ ファイルを IOS XR にコピーします。

GPB エンコーダの設定

以降に示すステップで概説するように、GPB エンコーダを設定し、テレメトリ ポリシーをアクティブ化してデータをストリーミングします。

1 次に示すように、ループバック インターフェイス アドレスを設定して IOS XR にテレメトリ レシーバをマッピングします。

```
RP/0/RP0/CPU0:ios(config)#interface Loopback2
RP/0/RP0/CPU0:ios(config-if)#ipv4 address 2.2.2.2/32
RP/0/RP0/CPU0:ios(config-if)#no shut
RP/0/RP0/CPU0:ios(config-if)#commit
Fri Oct 30 07:51:14.785 UTC
RP/0/RP0/CPU0:ios(config-if)#exit
RP/0/RP0/CPU0:ios(config)#exit
RP/0/RP0/CPU0:ios#show ipv4 interface brief
Fri Oct 30 07:51:48.996 UTC

Interface                      IP-Address      Status                Protocol
Loopback0                      1.1.1.1         Up                    Up
Loopback1                      8.8.8.8         Up                    Up
Loopback2                      2.2.2.2         Up                    Up
GigabitEthernet0/0/0/0         192.164.168.10  Up                    Up
GigabitEthernet0/0/0/1         192.57.43.10    Up                    Up
GigabitEthernet0/0/0/2         unassigned      Shutdown              Down
MgmtEth0/RP0/CPU0/0            192.168.122.197 Up                    Up
RP/0/RP0/CPU0:ios#
```

2 エンコーダを設定し、前のステップで設定した IOS XR のループバック インターフェイスにポリシーをストリーミングします。

```
telemetry
          encoder gpb
            policy group alpha
              policy demo
              destination ipv4 2.2.2.2 port 5555
            !
      !
   !
```

サードパーティ製コンテナ（LXC）の起動

ここでは、IOS XR のサードパーティ製コンテナ（LXC）を起動する方法について説明します。

1. IOS XR にログインし、デフォルトの名前空間（XRNNS）に移動します。

```
RP/0/RP0/CPU0:ios# run
[xr-vm_node0_RP0_CPU0:~]$
```

2. サードパーティ製コンテナを起動します。

```
[xr-vm_node0_RP0_CPU0:~]$virsh -c lxc+tcp://10.11.12.15:16509/ -e ^Q console demo1
```

3. 要求されたら、コンテナにログインします。

```
Connected to domain demo
Escape character is ^Q
Kernel 3.14.23-WR7.0.0.2_standard on an x86_64
host login: Password:
```

これで、サードパーティ製コンテナが正常に起動されました。

テレメトリ レシーバの設定

テレメトリ レシーバは、ストリーミングされたデータを指定したインターフェイスIPアドレスとポート番号でリッスンし、受信したパケットのヘッダーを印刷します。.proto ファイルが指定されている場合は、プロトコルコンパイラを使用してそのファイルがコンパイルされ、メッセージコンテンツも印刷されます。デフォルトでは、各テーブルの最初の行は印刷されますが、出力全体を印刷するには `print-all` オプションを使用します。

起動したコンテナ内でテレメトリ レシーバを実行するには、次のステップを実行します。

1. すべてのレシーバ ファイルをサードパーティ製コンテナにダウンロードします。https://github.com/cisco/bigmuddy-network-telemetry-collector では、レシーバ ファイルを IOS XR で使用できます。

2. レシーバを実行してデータを印刷します。

```
python gpb_receiver.py ipaddress 2.2.2.2 port 5555 proto
generic_counters.proto ipv4_counters.proto
```

次に示すように、テレメトリ レシーバにデータが表示されます。

```
Waiting for message
Got message of length:1036bytes from address:('10.1.1.1', 5555)
Encoding:2271560481
Policy Name:demo
Version:25
Identifier:<data that may be sent by the encoder to the mgmt stn>
Start Time:Wed Jan 21 09:54:33 1970
End Time:Wed Aug 26 09:28:37 2015
# Tables:1
Schema
Path:RootOper.InfraStatistics.Interface.Latest.GenericCounters
# Rows:6
Row 0:
applique:0
availability_flag:0
broadcast_packets_received:0
broadcast_packets_sent:0
bytes_received:0
bytes_sent:0
carrier_transitions:0
crc_errors:0
framing_errors_received:0
```

```
giant_packets_received:0
input_aborts:0
input_drops:0
input_errors:0
input_ignored_packets:0
input_overruns:0
input_queue_drops:0
interface_name:Null0
last_data_time:1440606516
last_discontinuity_time:1440498130
multicast_packets_received:0
multicast_packets_sent:0
output_buffer_failures:0
output_buffers_swapped_out:0
output_drops:0
output_errors:0
output_queue_drops:0
output_underruns:0
packets_received:0
packets_sent:0
parity_packets_received:0
resets:0
runt_packets_received:0
seconds_since_last_clear_counters:0
seconds_since_packet_received:4294967295
seconds_since_packet_sent:4294967295
throttled_packets_received:0
unknown_protocol_packets_received:0
Waiting for message
Got message of length:510bytes from address:('2.2.2.2', 5555)
Encoding:2271560481
Policy Name:demo
Version:25
Identifier:<data that may be sent by the encoder to the mgmt stn>
Start Time:Wed Jan 21 09:54:33 1970
End Time:Wed Aug 26 09:28:38 2015
# Tables:1
Schema Path:RootOper.InfraStatistics.Interface.Latest.Protocol
# Rows:5
Row 0:
bytes_received:0
bytes_sent:0
input_data_rate:0
input_packet_rate:0
interface_name:Loopback2
last_data_time:1440606517
output_data_rate:0
output_packet_rate:0
packets_received:0
packets_sent:0
protocol:24
protocol_name:IPV4_UNICAST
```

テレメトリ レシーバはサードパーティ製コンテナ（LXC）内で正常に動作します。