

ベンダNOSのOpen化とその活用事例、 今後の展望

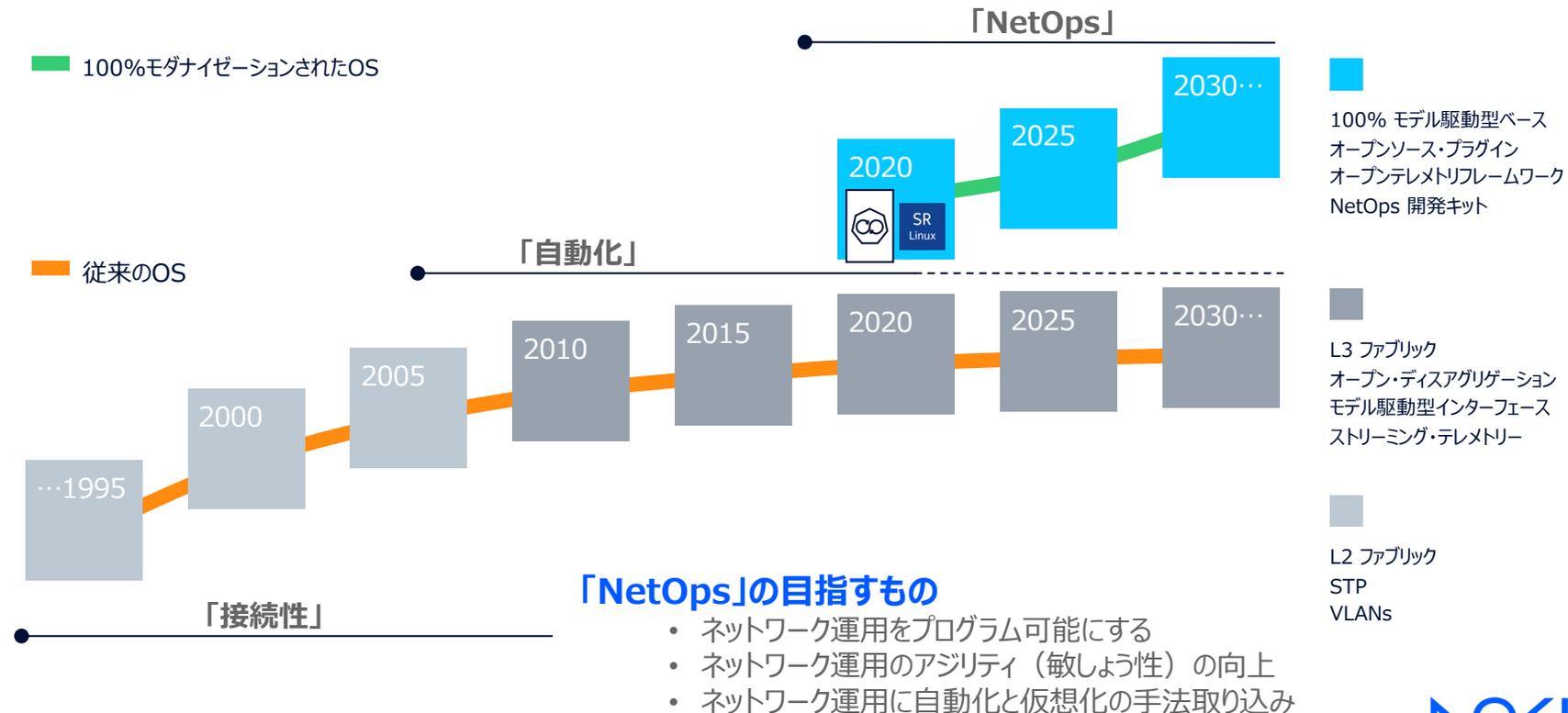
鹿志村 康生

yasuo.Kashimura@nokia.com

NOKIA

ネットワークオペレーションの変化

NetOps = 次世代NOSの必要性



NetOps の目的

1. プログラム化・オープン化によるネットワーク設計運用の強化

- プログラム可能なネットワークにより、ソフトウェアを使用した構成と管理の簡素化が可能。
- NW機器の設定や変更のコード化により一貫性が確保され、作業ミスが回避される。
- オープンAPIを活用することで、他のシステムとの統合におけるコストが削減される。

2. アジリティ（敏しように性）の向上によるネットワーク設計運用の進化

- ビジネス要求の変化に迅速に対応できる柔軟性が実現。
- 適切な自動化により、デプロイメントや変更の速度が大幅に向上。
- システム全体の効果的な監視と分析が、問題の早期発見と迅速な対応を可能にする。

3. 自動化と仮想化の導入によるネットワーク設計運用の最適化

- ルーチンタスクの自動化により、手作業エラーの減少と作業効率の向上。
- サービス展開の迅速化によりビジネスチャンスの拡大。
- NFVやコンテナ等の仮想化技術により、ハードウェアへの依存を削減し、検証から商用環境への移行を効率的に実施。

最新のソフトウェアアーキテクチャーに基づいた新しいソリューションの必要性

Nokia SR Linux NOS

柔軟でオープンなアプリケーション開発プラットフォーム

オペレーター
アプリケーション

ネットワーク
プローブ

モニタリング

デバイス設定

デバイス
バックアップ

ネットワーク
ヘルス

柔軟な運用

オープンで拡張可能
テレメトリ
フレームワーク



オープンソース
CLIプラグイン
(Python)



NetOps
開発
キット (NDK)



強固な基盤

スタンダード
Linux
カーネル

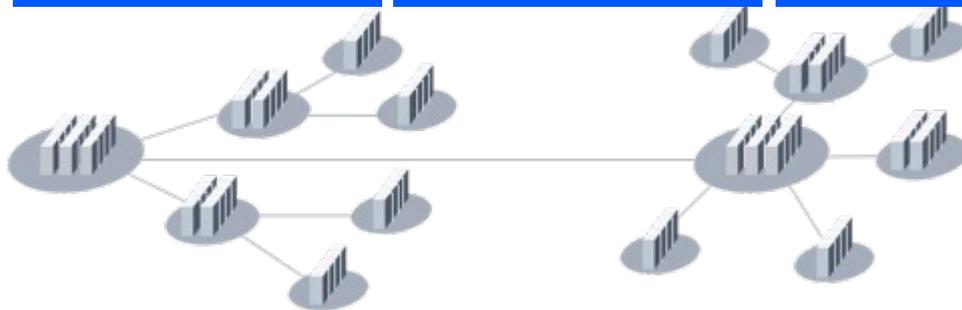


グラウンドアップ
モデルドリブン
ファウンデーション



耐障害性
実績のある
プロトコルスタック

SR
OS



Nokia SR Linux NOS

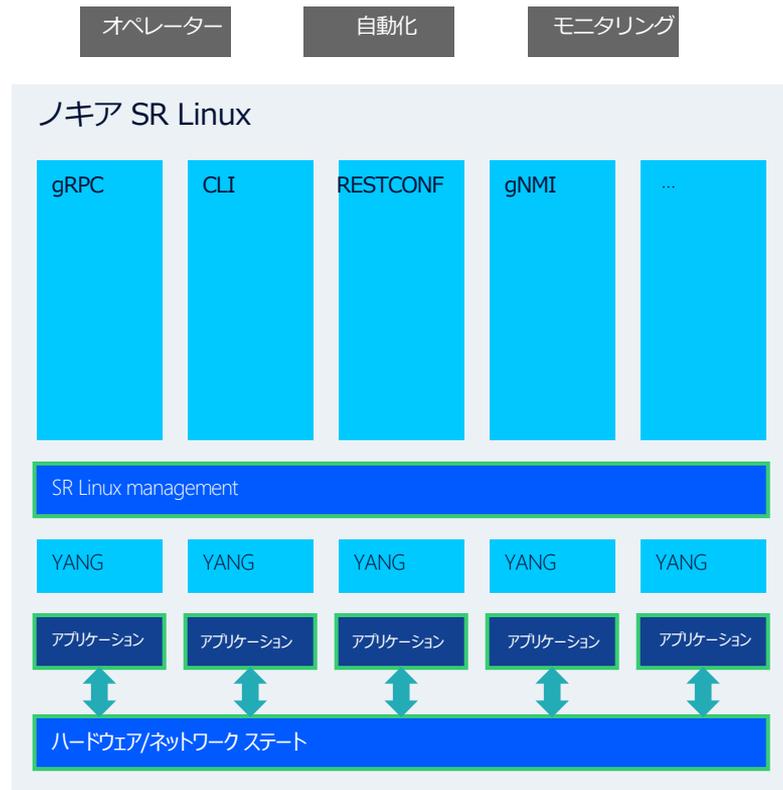
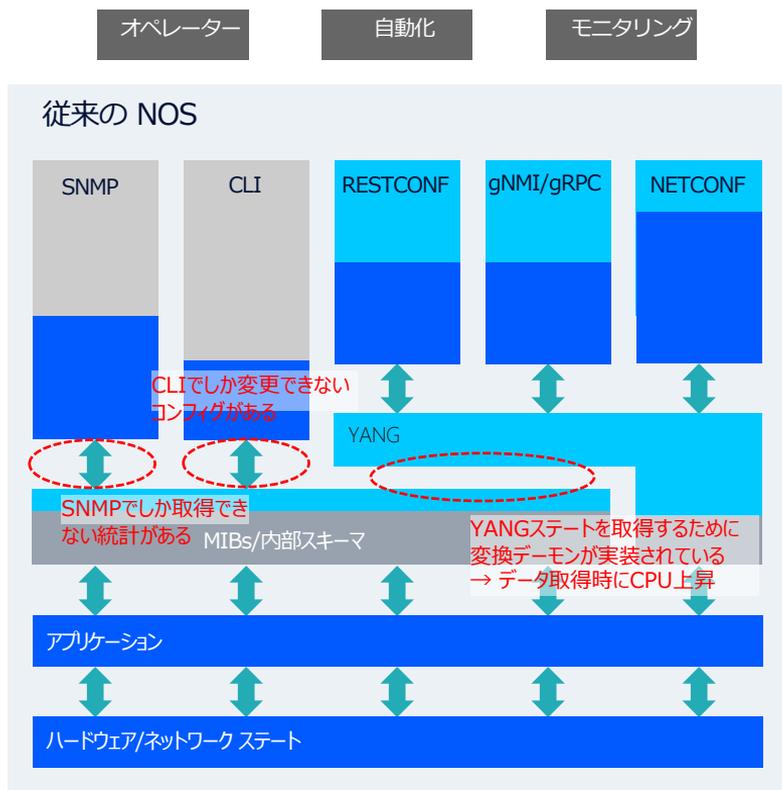
従来のNOSの課題・制約をクリアする

	ハードウェアとネットワークOSを別々に組み合わせるアプローチ	ハードウェアとネットワークOSを一括で購入するアプローチ
個別のインテグレーション	必要	不要
保守の複雑さ(切り分け等)	高い	低い
機能の柔軟性	高い	低い
カスタマイズ性	高い	低い
技術的なチャレンジ	高い	低い

“オープン化”によりここを改善

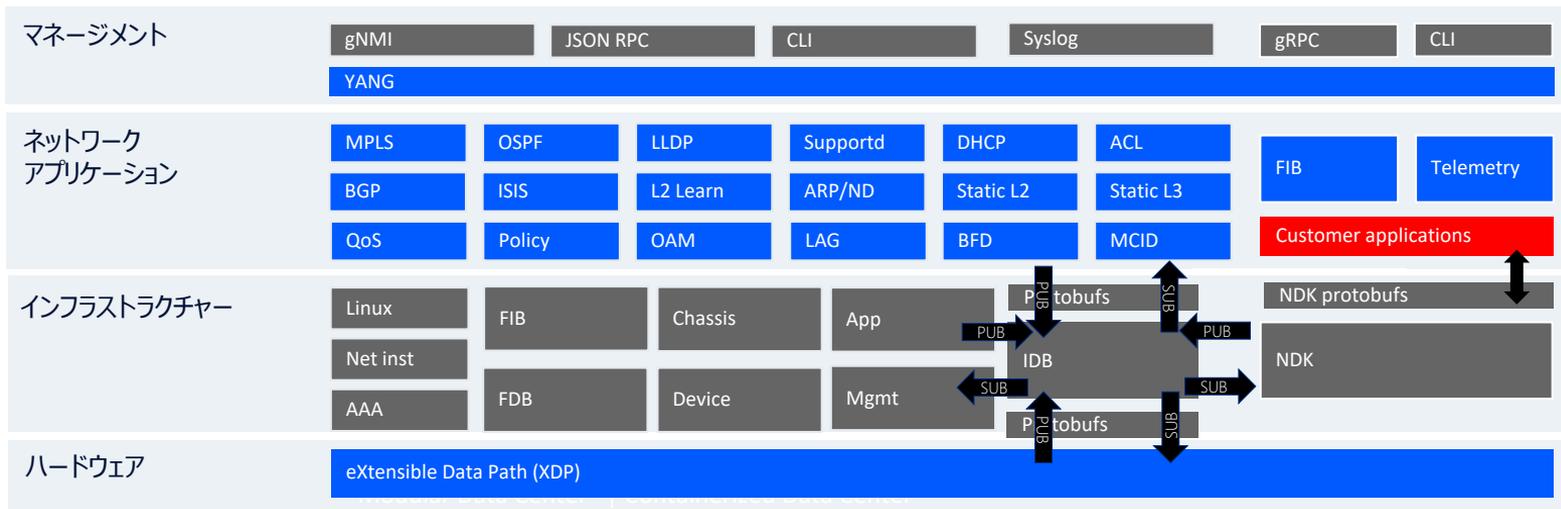
Nokia SR Linux NOS

従来のNOSの課題・制約をクリアする



Nokia SR Linux NOS アーキテクチャ

完全にオープンなインフラ



実績のあるIPプロトコルスタック

- 20年以上にわたり実績のある安定した Nokia SR OSのプロトコルスタックとネットワークサービスのフィールドハードニング



モデル駆動型マネジメント

- テレメトリーによる柔軟で統一されたプログラマビリティの実現
- OpenConfig及びNativeモデルのサポート

オープン・インフラ

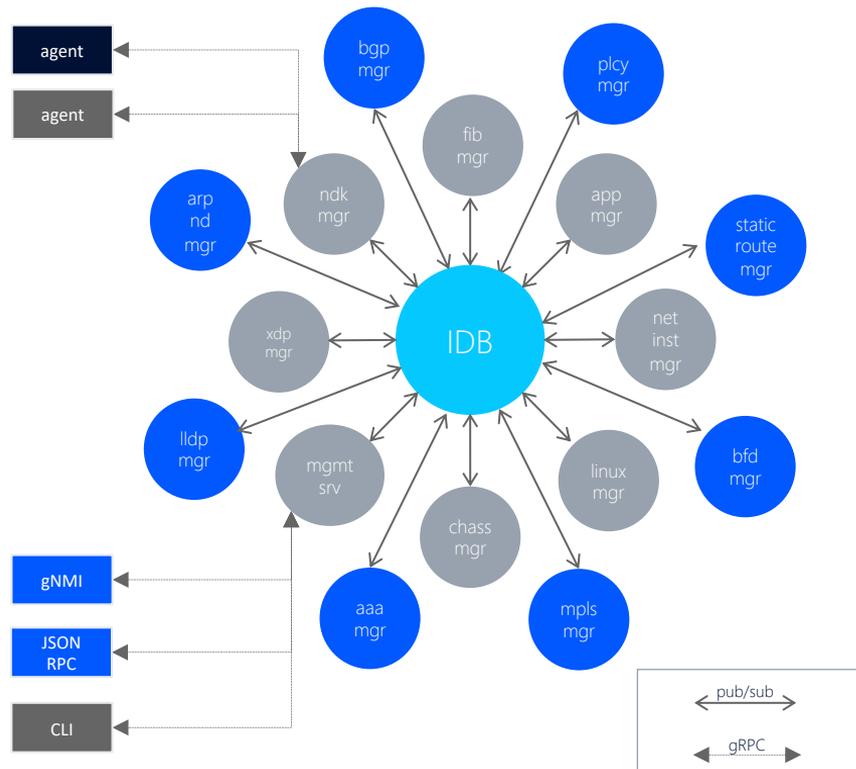
- NDKによるNativeなgRPC/protoエクステンション
- プラグインによるPython CLIの拡張性
- uPythonによるテレメトリーベースイベントハンドラー
- NativeなCustom applicationの統合

NOKIA

Nokia SR Linux NOS アーキテクチャ

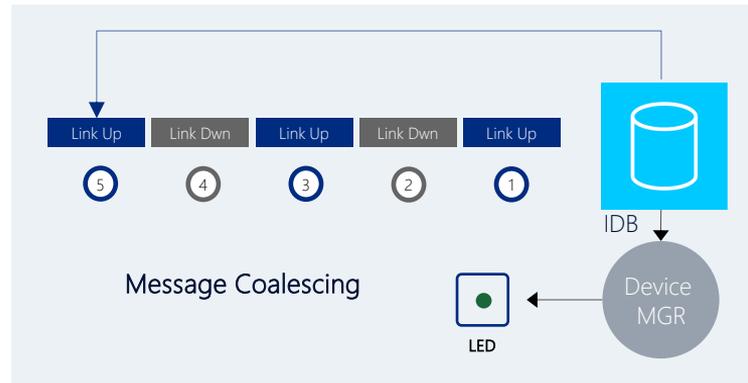
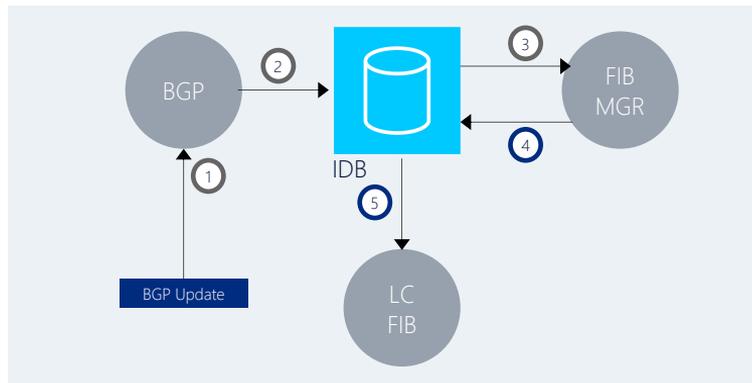
IDB: インポートデータベース

- 軽量なデータベース (redisスタイル、Pub/Subベース)
- DBはトピックに分割される
- 各トピックは1つのアプリケーションによって書き込まれる。
(複数トピックの書き込みは不可)
- アプリケーションはIDBへのセッションを開いたときにトピックを購読し、他のアプリケーションが利用できるように自分のトピックにメッセージを発行。
- エージェントがシステム内の動作を直接操作できるようにする。
例えば、FIBマネージャが選択するために独自のルートを公開することが可能。
- アプリケーションの状態はアプリケーション内に存在し、他のアプリケーションが関心を持つ状態のみがIDBに公開される。
- IDB を合理化し、不必要な肥大化を排除、IDB はデータのペイロードや構造を意識する必要がない
- テレメトリーは常にアプリケーションから直接ストリームされる。
- データベースに永続性がないため、データストアというよりもキャッシュメッセージベースとして機能し、スピードを向上。
- 強力なNDKとともに提供され、オペレータはシステム内のイベントに基づいて外部イベントハンドリングを行うことが可能。



Nokia SR Linux NOS アーキテクチャ

IDB: インポートデータベース



状態共有/IDBアーキテクチャ

- アプリケーションが停止/再起動/故障した場合、アプリケーションの公開データはパージされる
- サブスクライブするアプリケーションは、公開された更新をリッスンし、独立して反応
- アプリケーションの速度が遅い場合、そのアプリケーションだけが遅れ、他のシステムは正常に機能する
- アプリケーションの再起動のためのタイマーを保持する (Warmリロード/ヒットレスアップグレード/ISSU)

インテリジェントな効率性

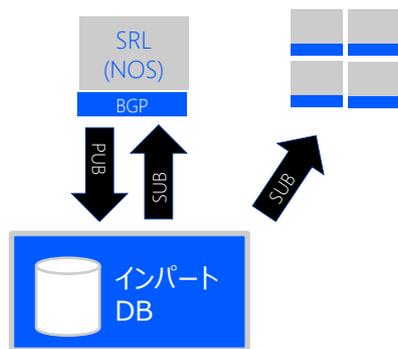
最新の情報を提供するために、メッセージを統合する機能を持つ。

1. 緑色のLEDの更新 Link Status = Up
2. IDBが発行する複数のリンクダウン、アップメッセージ(1,2,3,4,5)
3. IDBは、すべてのアプリケーションにすべてのメッセージを処理させるのではなく、インテリジェントに最新の公開メッセージを合体して渡します。
4. アプリケーションの処理が追いついている場合は必要ない

Nokia SR Linux NOS アーキテクチャ

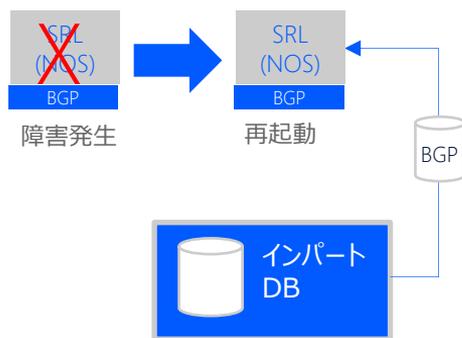
IDBで実現する耐障害性の向上

正常状態



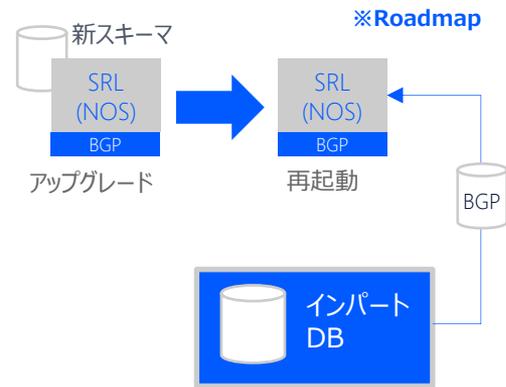
ネットワークアプリは、データをIDBに書き込み。他のアプリはのIDBからデータを読み込む。

NWアプリレベルの障害



障害が発生した場合、アプリは再起動される。IDBに公開されたデータは保持され、データを再取得することでサービス継続が可能になる。

NWアプリレベルのファイル更新



アップグレード時にアプリを再起動する。新しいスキーマがロードされ、IDBに公開されていたデータは保持されるので、データ同期とサービスの継続が可能となる。

Nokia SR Linux NOS アーキテクチャ

ツール群 – 多面的な管理インフラをサポート



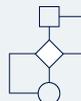
Auto-boot



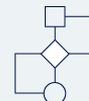
DevOps CLI



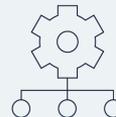
Telemetry



gNMI



JSON RPC



NDK

ZTP

- カスタマイズ可能な Python実装
- v4/v6
- HTTP/s
- アウトオブバンド & インバンド

ONIE

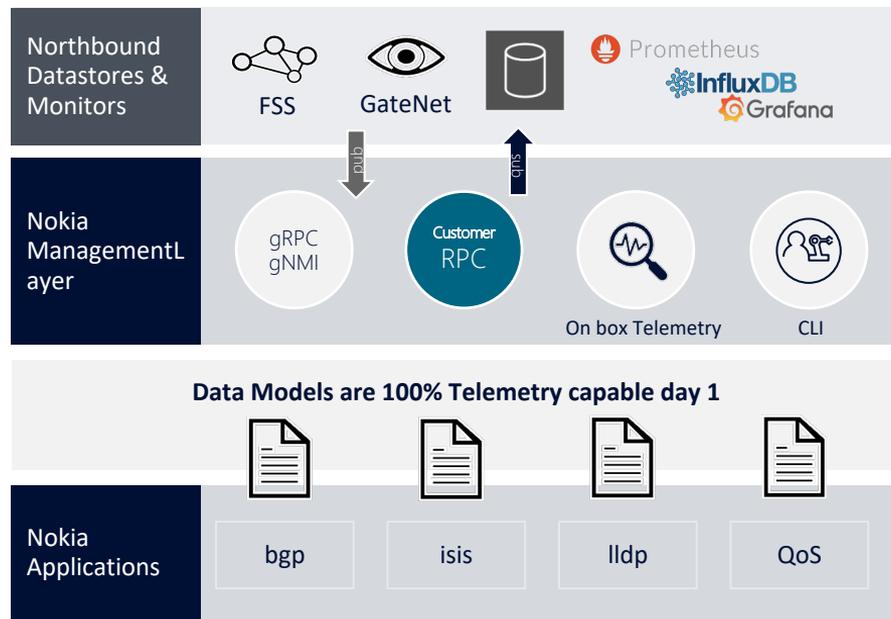
- Pythonベース、完全なオープンソース
- プラグインによる無制限の拡張が可能
- エイリアス、ユーザーごとの環境
- ネイティブな Telemetry アーキテクチャ
- スケーラブルで堅牢
- オートテレメトリフレームワーク
- Googleが選んだストリーミングインタフェース
- gRPC/プロトベース
- 設定とテレメトリのための統一された堅牢な管理用プロトコル
- 設定と状態の取得のためのプログラム可能なJSONインターフェース
- CLICコマンドのサポート
- Nokiaの拡張機能 (commit confirmed, validateなど) を使用可能
- NetOps Development Kitは、SR Linuxのインフラをサービスとして公開
- アプリケーションは、トランザクション設定、ストリーミング・テレメトリを利用可能

Nokia SR Linux NOS

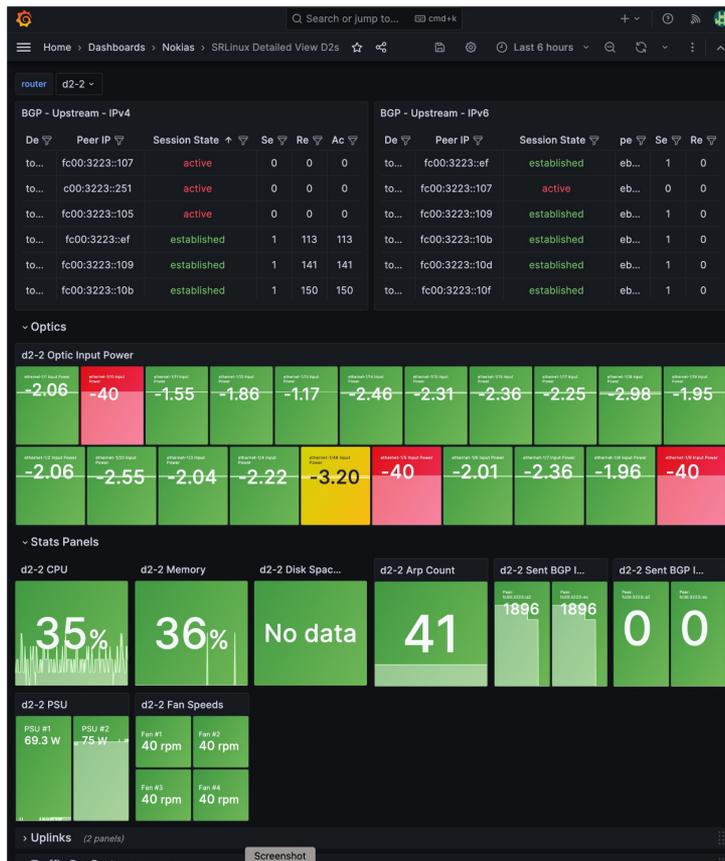
ストリーミングテレメトリー

モデル・ドリブン・テレメトリー

- テレメトリーにより、リアルタイム性の高いより良いネットワーク管理を実現可能。**任意の YANG パス**にサブスクライブ。
- jsonとprotoを用いた**標準ベースの gNMI/gRPC**を活用した**完全な相互運用性**（プラットフォームやベンダーに非依存）。
- モデルインフラは、**オンボックスおよびオフボックスのテレメトリーをサポート**



テレメトリを活用したユーザーダッシュボード お客様での作成事例



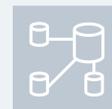
Nokia SR Linux NOS アーキテクチャ マネージメントAPI

管理インターフェース – SR-Linux CLI

- CLIの完全なグランドアップ設計、業界全体のペインポイントや欠陥の認識
 - オペレータにツールを提供し、オペレータが望むものを見ることができるようにするコンセプトで開発
- CLIは、管理サーバーと対話するためにgRPCを使用するクライアント
 - 全てPythonで記述され、オープンソースで、プラグインが可能
- CLIは、SRLinuxモデル駆動アーキテクチャを活用 - CLIツリーは、SRLinuxとオペレータのYANGモジュール、エイリアス、プラグインから派生
- 高い柔軟性
 - 監視コマンド - 変更サブスクリプションのgNMIを介して「任意の」リーフを監視する機能
 - グローバル/ルートの全置換
 - リーフやコンテナなど、どこでもアノテーションが可能
 - 設定と状態のJSON出力
 - 複数の出力修飾子、リダイレクト
 - 範囲指定、ワイルドカードのサポート
 - エイリアス、環境変数
 - コンテキストの認識（ネットワークインスタンス、ショールーチン）
 - コマンドラインのオートコンプリート/オプションのプレビュー - インテリジェントなエンジンにより、YANG、プラグイン、またはエイリアスによる補完が可能

マネージメント/オペレーション

CLI



Management server

YANG

Nokia

Customer

アプリケーション

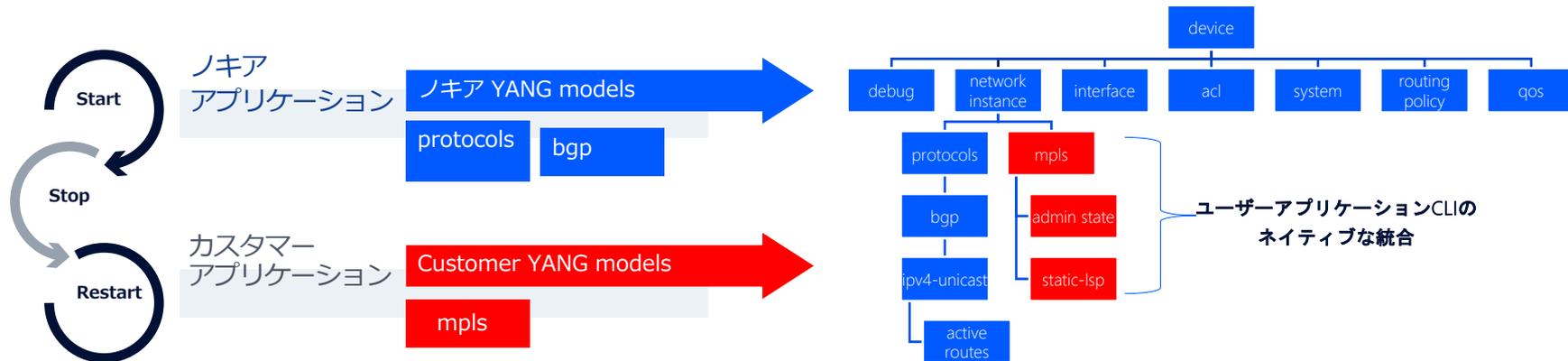
SR Linux
Apps

3rd party
Apps/ Agent

Nokia SR Linux NOS

CLIカスタマイゼーション

- 従来のCLIに加えてオペレータ独自のCLIを定義可能。保守運用のオペレーションコスト、自動化ツールのコスト削減を実現します。



ライフサイクルマネジメント

カスタマーアプリケーションは最重要なものとして管理され、耐障害性やオンボーディングのシンプル化を提供

YANG インテグレーション

完全なYANG IntegrationによるネイティブCLI, gNMI, JSON-RPC – 永続的なconfig, ステート, スタッツ, Show コマンド
任意のオブジェクト、任意のインタフェース

PythonベースのCLI

複数のYANGモデル/パスからのユーザ定義アウトプットを使用してカスタムプラグインを作成する
完全なPython実装、オープンソース、高い拡張性

Nokia SR Linux NOS

カスタムCLIの事例 1

- オペレータ固有構成においてLeafスイッチ運用で使用する出力を1つのCLIで実装

```
class Plugin(CliPlugin):  
    def load(self, cli, ** kwargs):  
  
    def _show_help (self, state, output, **_kwargs):  
        print('''  
The 'show fabric' command shows you statistics and the status of the uplinks and BGP peerings.  
Therefore it requires some inputs that need to be added in the 'fabric.py' file.  
  
'/opt/srlinux/python/virtual-env/lib/python3.6/site-packages/srlinux/mgmt/cli/plugins/reports/fabric.py'  
  
Example:  
interfaces = []          # fill the interfaces  
description = 'spine'  
uplink_peer_group = 'spine'  
rr_peer_group = 'EVPN-NVO'  
''')  
  
    def _show_summary(self, state, output, **_kwargs):  
  
        header = f'Fabric Connectivity Report'  
        result_header = self._populate_header(header)  
        self._set_formatters_header(result_header)  
        output.print_data(result_header)  
        self._show_uplinks(state, output)  
        self._show_rr(state, output)  
        self._show_stats(state, output)
```

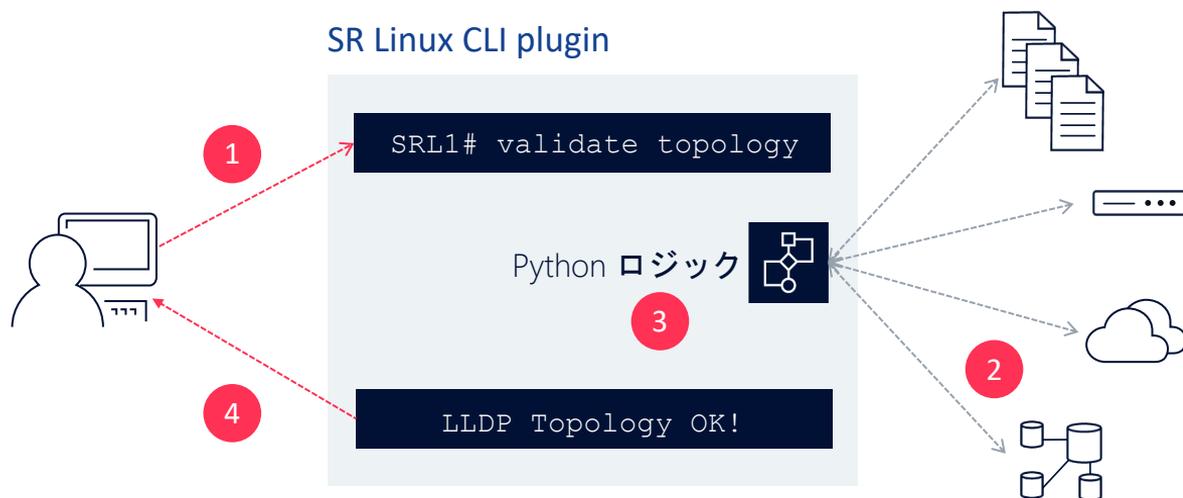
Spineとの接続Port/BGPの情報を一括で表示

Fabric接続ポートのみの情報を表示

```
--{ running }--[ ]--  
A:dcs1-clab-fab1p1-leaf2# show fabric uplink  
-----  
Local Interface | Local Router | Link Status | eBGP Status | Remote Router | Remote Interface  
-----  
ethernet-1/1 | dcs1-clab-fab1p1-leaf2 | up | established | dcs1-clab-fab1p1-spine1 | ethernet-1/2  
ethernet-1/2 | dcs1-clab-fab1p1-leaf2 | up | established | dcs1-clab-fab1p1-spine2 | ethernet-1/2  
-----  
  
--{ running }--[ ]--  
A:dcs1-clab-fab1p1-leaf2# show fabric route-reflector  
-----  
Route Reflector Address | iBGP Status | Neighbor Description | Rx/Active/Tx | Uptime (hh:mm:ss) |  
-----  
fd00:feed:beef:0:1:1:1:1 | established | BGP-EVPN to RR | 22/14/10 | 1 day, 5:19:50 |  
fd00:feed:beef:0:1:1:2:1 | established | BGP-EVPN to RR | 22/0/10 | 1 day, 5:19:52 |  
-----  
  
--{ running }--[ ]--  
A:dcs1-clab-fab1p1-leaf2# show fabric statistics  
-----  
Local Interface | Traffic Bps In/Out | Packets In/Out | Errored In/Out | FCS Err | CRC Err | Transceiver Volt |  
-----  
ethernet-1/1 | 321/321 | 14211/14204 | 0/0 | 0 | 0 | N/A  
ethernet-1/2 | 68/68 | 14216/14214 | 0/0 | 0 | 0 | N/A  
-----  
  
--{ running }--[ ]--  
A:dcs1-clab-fab1p1-leaf2#
```

Nokia SR Linux NOS

カスタムCLIの事例 2



1. カスタムCLIコマンドを実行
2. CLIプラグインがトポロジーDBから接続しているべきLLDP Neighborの情報を取得
3. DBから取得したLLDP Neighborの情報とLocalのLLDP Neighbor stateを比較
4. 結果をOutputとして表示

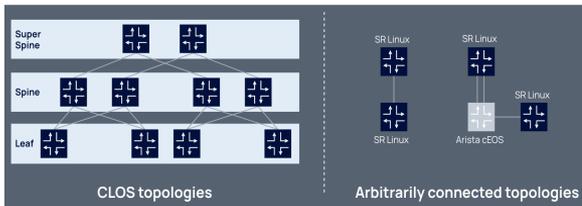
オープンな開発／テスト環境へのコントリビューション

オープンに活用できるツール群

- SR-Linuxはリリースされると Git Hub のオープンネットワークに公開
- ライセンスなしで商用イメージと同じ機能をすぐに検証可能
- オープンなコンテナラボを利用することでノートPCでもラボ環境を構築可能

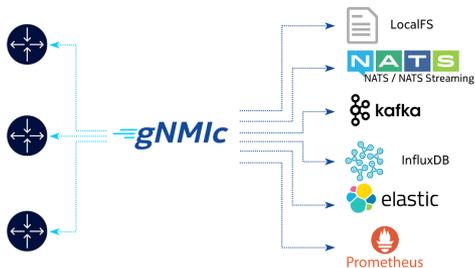
<https://containerlab.dev/>

<https://www.youtube.com/watch?v=xdi7rwdJgkg>



release **v0.12.0** downloads **1.2k**

<https://gnmic.kmr.d.dev/>



→ gNMI subscription
→ JSON/Proto/Event

<https://learn.srlinux.dev/>

<https://github.com/srl-labs/containerlab>



Discord

<https://discord.gg/vAyddtaEV9>



Containerlabのメリット



サポート済みNOS

NOKIA

srl
vr-sros

JUNIPER
NETWORKS

crpd
vr-vmx
vr-vqfx

ARISTA

ceos
vr-veos

CISCO

vr-xrv9k
vr-csr
vr-n9kv

NVIDIA

CVX

SONIC

sonic-vs
fr

paloalto
NETWORKS

vr-pan

DELL

vr-ftosv

ixia

Keysight_ixia-c

ipinfusion

ipinfusion_ocnos

MikroTik

vr-ros

Containerlabのメリット

様々なラボ構成例を公開中

The screenshot displays the Containerlab web interface for a "5-stage Clos topology". The interface includes a search bar, a user profile for "srl-labs/containerlab", and a table of contents with links for "Description" and "Configuration".

The central diagram illustrates a multi-stage Clos network topology. At the top, there are J Super-Spines (represented by dark blue circles). Below them are N Spines (represented by blue circles), which are fully connected to the Super-Spines. At the bottom, there are M Leaves (represented by light blue circles), which are fully connected to the Spines. The entire structure is organized into K Pods. Brackets and labels indicate the hierarchical structure: J Super-Spines, N Spines, M Leaves, and K Pods.

On the left side, a list of "Lab examples" is provided:

- 3-nodes Clos fabric
- 5-stage Clos fabric
- 5-stage SR Linux based Clos fabric with config engine
- Nokia SR Linux and Arista cEOS
- Nokia SR Linux and Juniper cRPD
- Nokia SR Linux and SONiC
- External bridge capability
- WAN topology
- Nokia SR Linux and Nokia SR OS
- Nokia SR Linux and Juniper vMX
- Nokia SR Linux and Cisco XRv9k
- Nokia SR Linux and Cisco XRv
- Nokia SR Linux and FRR
- FRR
- Cumulus Linux and FRR
- Cumulus Linux (docker runtime) and Host
- BGP VPLS between Nokia and Juniper
- Keysight IXIA-C and Nokia SR Linux

At the bottom, the "Configuration" section shows a terminal window with the command:

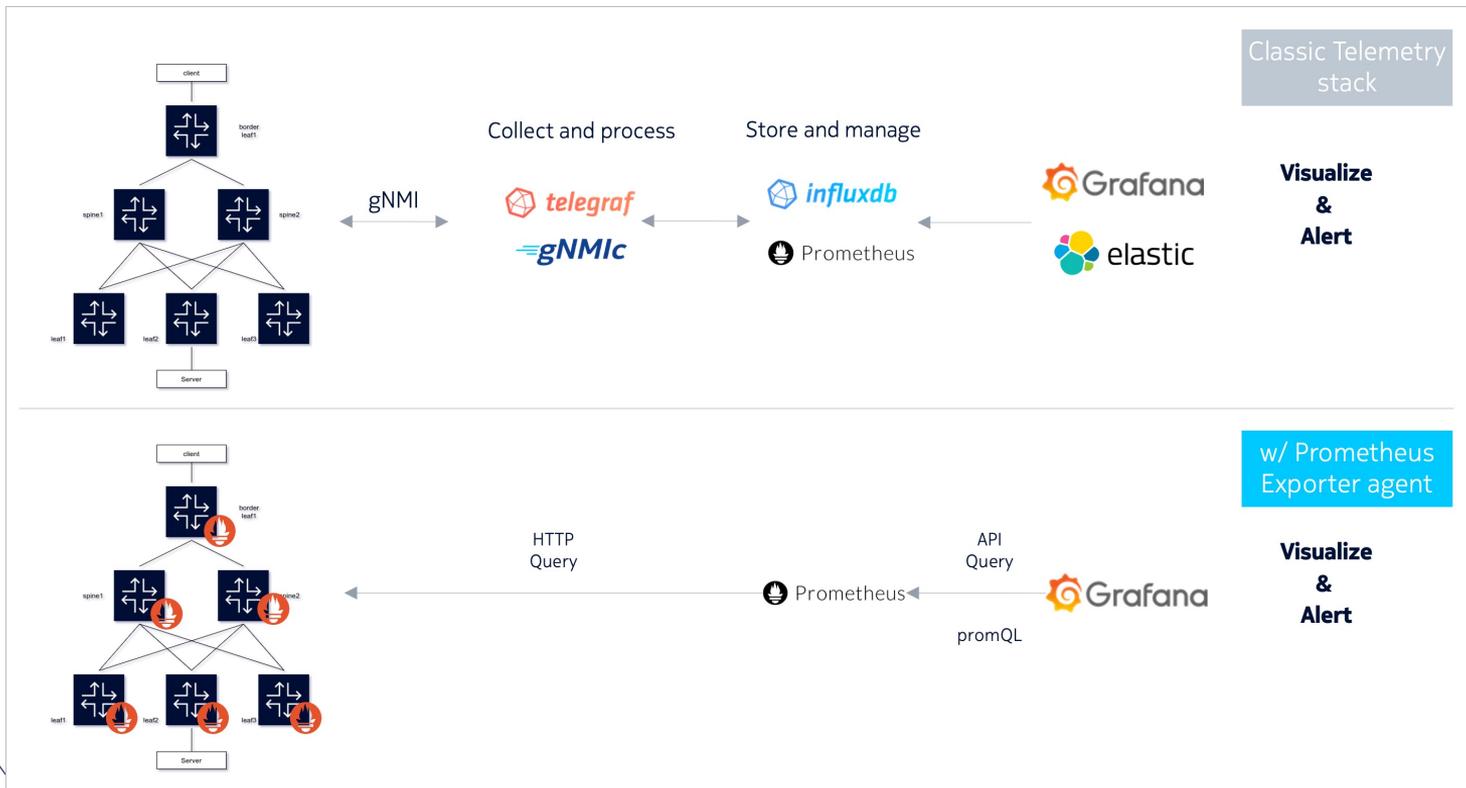
```
clab deploy -t templated02.clab.gotmpl
```

Below the terminal, a note says: "Run configure.sh script to configure the lab".

NDKを活用したカスタマーアプリケーション開発

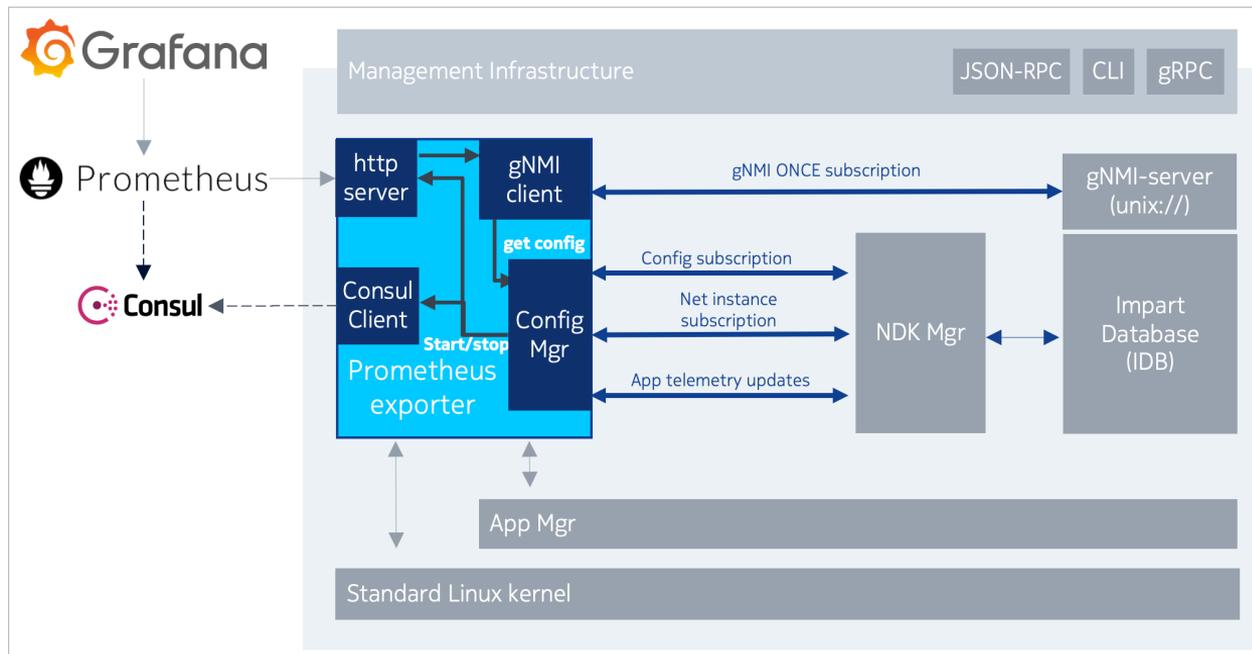
ユースケース 1 : Prometheus Exporter Agent

Prometheus Export AgentをSR Linuxのアプリケーションとして導入
→ 可視化構成の簡素化を実現



NDKを活用したカスタマーアプリケーション開発

ユースケース1 : Prometheus Exporter Agent

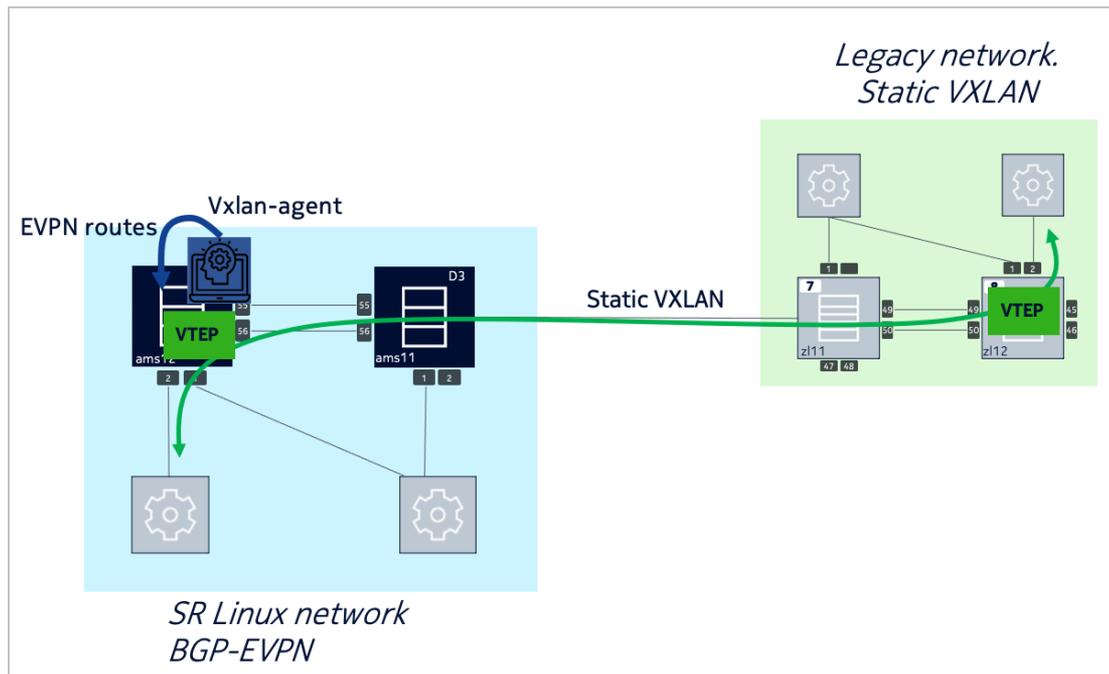


<https://learn.srlinux.dev/ndk/apps/srl-prom-exporter/>

NDKを活用したカスタマーアプリケーション開発

ユースケース2 : EVPN Proxy Agent

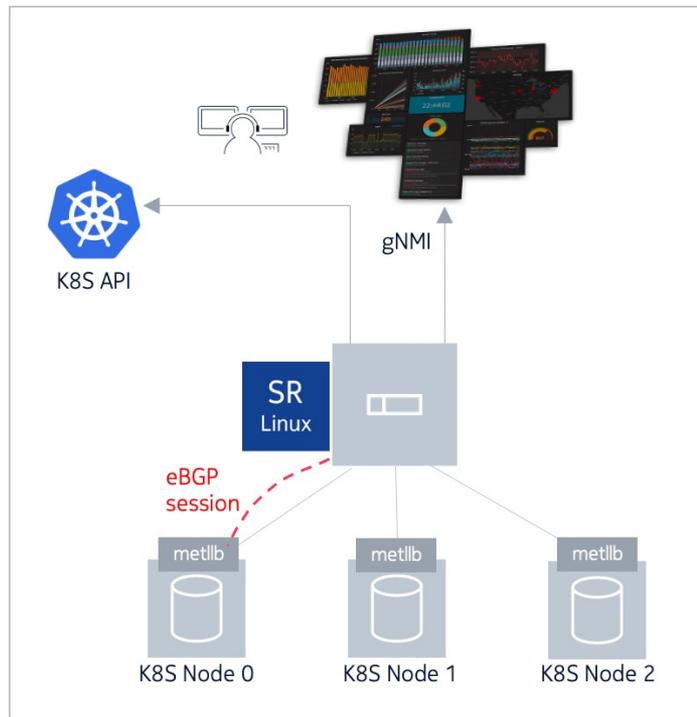
エージェントはSR Linux NOSにインストールされ、レガシーVTEPデバイスに代わってEVPNルートを生成することにより、スタティックVXLAN VTEPとEVPN対応サービス間のコントロールプレーンステッチングを可能とする

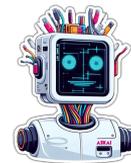


NDKを活用したカスタマーアプリケーション開発

ユースケース 3 : kButler

- K8s仮想基盤の各Podが保持している経路情報が想定した状態かどうかをSR Linux(リーフスイッチ)から監視
- トラフィック状態をテレメトリで監視。トラフィックの送受信がない場合にはgNMIにて監視システムへ通知





NDKを活用したカスタマーアプリケーション開発

Experimental / SR Linux上でのChatGPTを使用したAIエージェント

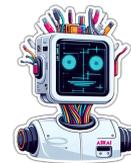
<https://learn.srlinux.dev/blog/2023/sr-linux-gpt/>

```
root@AF01-003: ~
--[ + running ]--[ ]--
A:srl# askai can you tell me how to configure EVPN instance ?
To configure an EVPN instance, you need to set up a vxlan-interface, a MAC-VRF network-instance, and a BGP-EVPN configuration.
Here is a basic EVPN-VXLAN L2 configuration example:

First, create a tunnel-interface and vxlan-interface:
...
set / tunnel-interface vxlan1
set / tunnel-interface vxlan1 vxlan-interface 1
set / tunnel-interface vxlan1 vxlan-interface 1 type bridged
set / tunnel-interface vxlan1 vxlan-interface 1 ingress
set / tunnel-interface vxlan1 vxlan-interface 1 ingress vni 10
set / tunnel-interface vxlan1 vxlan-interface 1 egress
set / tunnel-interface vxlan1 vxlan-interface 1 egress source-ip use-system-ipv4-address
...

Then, associate the vxlan-interface to the network-instance and associate the vxlan-interface to the bgp-evpn instance:
...
set / network-instance blue
set / network-instance blue type mac-vrf
set / network-instance blue admin-state enable
set / network-instance blue description "Blue network instance"
set / network-instance blue interface ethernet-1/2.1
set / network-instance blue vxlan-interface vxlan1.1
set / network-instance blue protocols
set / network-instance blue protocols bgp-evpn
set / network-instance blue protocols bgp-evpn bgp-instance 1
set / network-instance blue protocols bgp-evpn bgp-instance 1 admin-state enable
set / network-instance blue protocols bgp-evpn bgp-instance 1 vxlan-interface vxlan1.1
set / network-instance blue protocols bgp-evpn bgp-instance 1 evi 10
set / network-instance blue protocols bgp-vpn
set / network-instance blue protocols bgp-vpn bgp-instance 1
set / network-instance blue protocols bgp-vpn bgp-instance 1 export-policy pol-def-1
set / network-instance blue protocols bgp-vpn bgp-instance 1 import-policy pol-def-1
set / network-instance blue protocols bgp-vpn bgp-instance 1 route-distinguisher
set / network-instance blue protocols bgp-vpn bgp-instance 1 route-distinguisher rd 64490:200
set / network-instance blue protocols bgp-vpn bgp-instance 1 route-target
set / network-instance blue protocols bgp-vpn bgp-instance 1 route-target export-rt target

--[ + running ]--[ ]--
A:srl#
```



NDKを活用したカスタマーアプリケーション開発

Experimental / SR Linux上でのChatGPTを使用したAIエージェント

<https://learn.srlinux.dev/blog/2023/sr-linux-gpt/>

```
root@AF01-003: ~  
--[ + running ]--  
A:srl# askai list all ip routes with it's next hop  
Here are the IP routes with their next-hop interfaces:  
  
In the 'default' network instance:  
- Route to 192.168.1.0/30 via ethernet-1/1.0  
- Route to 192.168.1.2/32, local route, no next-hop interface  
- Route to 192.168.1.3/32, local route, no next-hop interface  
  
In the 'mgmt' network instance:  
- Default route (0.0.0.0/0) via mgmt0.0  
- Route to 172.20.20.0/24 via mgmt0.0, inactive  
- Route to 172.20.20.0/24 via mgmt0.0  
- Route to 172.20.20.13/32, local route, no next-hop interface  
- Route to 172.20.20.255/32, local route, no next-hop interface  
  
--[ + running ]--  
A:srl#
```


NOKIA