JANOG54講演

【講演資料】OpenConfigを活用したNW検証業務自動化への取り組みと課題

KDDI株式会社

佐藤 亮平
福井 良平

2024年7月05日

Tomorrow, Together KDDI おもしろいほうの未来へ。au

自己紹介

佐藤　亮平
Ryohei Sato

福井　良平
Ryohei Fukui

所属	IPネットワーク部 兼 次世代ネットワーク整備室	次世代ネットワーク整備室
経歴	◻ NW運用(2013～2018) AS2516網の運用・保守 ◻ NW設計・開発(2018～) ピアリング、将来NW検討、 自動化、RPKI、CONECT	◻ NW運用(2015～2019) FTTH網の運用・保守 ◻ NW設計・開発(2019～2021) AS2516網の開発 ◻ NW企画・戦略(2021～) 将来NW検討
JANOG参加歴	42@三重、43@山梨、 44@神戸、45@札幌 今回、初登壇	53@福岡 今回、初登壇

© 2024 KDDI

アジェンダ

- 背景・課題
- 課題への対応
- OpenConfig
- 検証自動化フレームワーク
- 技術実証（PoC）
- 今後の課題
- まとめ・議論したいポイント

どうしてOpenConfig？

そもそもOpenConfigって？

OpenConfigって使えるの？

© 2024 KDDI

NW検証業務の効率化が必要

- 通信NWが想定通り安全に動作・機能をすることを確認するため、数多くのNW検証試験を実施
- NW検証試験には、下記の課題がある

検証工数の削減

- **膨大な検証項目が存在**
  （OS Vup時の検証項目数は1500以上）
- **マルチベンダー・異なるOSの機器が存在**

商用適用へのリードタイム削減

- 検証には、**最低でも数か月要する**
- 想定外動作を確認した際には、切り分けやWA確立、場合によっては別OS Verへの切替が必要となり、商用適用が更に遅れる

手動での検証は、やってられない！

© 2024 KDDI

自動化・OpenConfig活用による効率化

■ NW検証業務自動化による効率化に加え、

OpenConfigを活用してマルチベンダー機器の操作をより効率的に行う

自動化 × **OpenConfigの活用**

- NW検証業務の**自動化により工数削減**
- ただし、様々なベンダ機器/OSに対応した設定・確認方法の定義やコーディングは大変

- 共通データモデルである**「OpenConfig」を活用し、効率的に自動化**を行う
- それに併せて**APIを利用**し、CLI操作やコマンド結果のparseから脱却する

検証			作業
基本機能検証	複合検証（システム検証）	作業手順検証	商用作業

今回のスコープ　　OpenConfigの活用は今後検討

© 2024 KDDI

OpenConfig

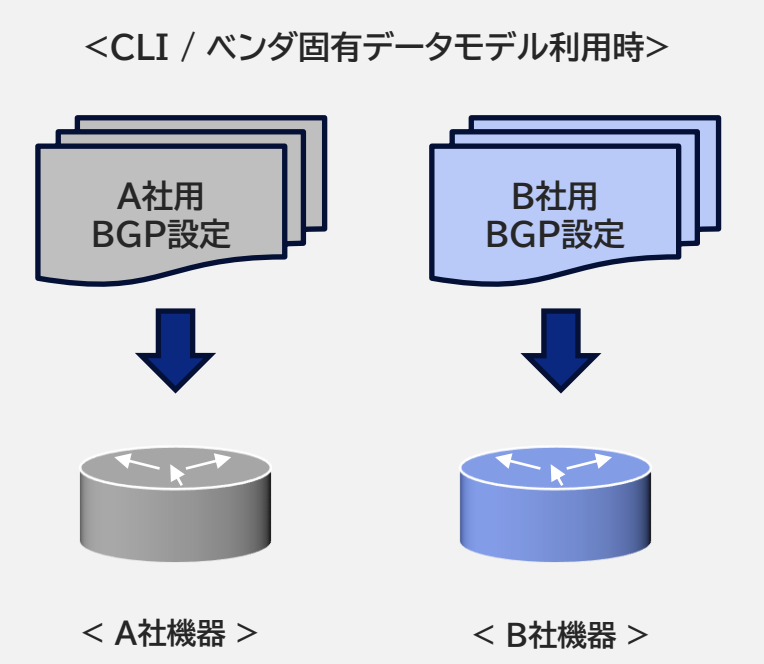
OpenConfig概要

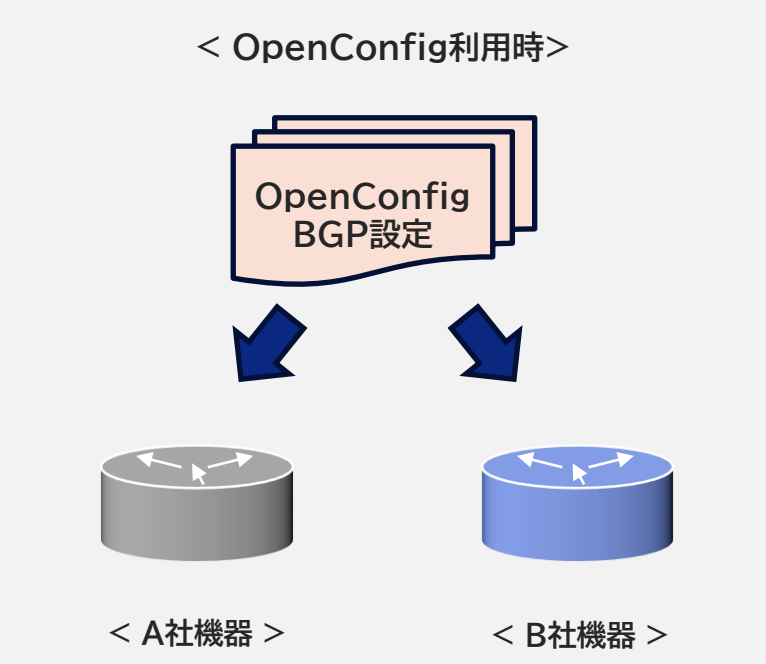
OpenConfigとは？

- 各ベンダ装置を一貫した手法で管理するための共通データモデル
- オペレーター主導で、WGにて共通データモデルを定義。既に各ベンダ機器で実装も進んでいる。

共通データモデル（OpenConfig）

従来は、機器/OSごとに異なるコマンドを利用したオペレーションが必要であったが、OpenConfigにより、共通のデータモデルに基づいて**統一したオペレーションが可能**となる





ベンダ実装状況

ルーティング／スイッチング/光トランスポート等の領域において、**主要ベンダの多くのプラットフォームで実装**されている

< OpenConfig対応ベンダ（一部）>

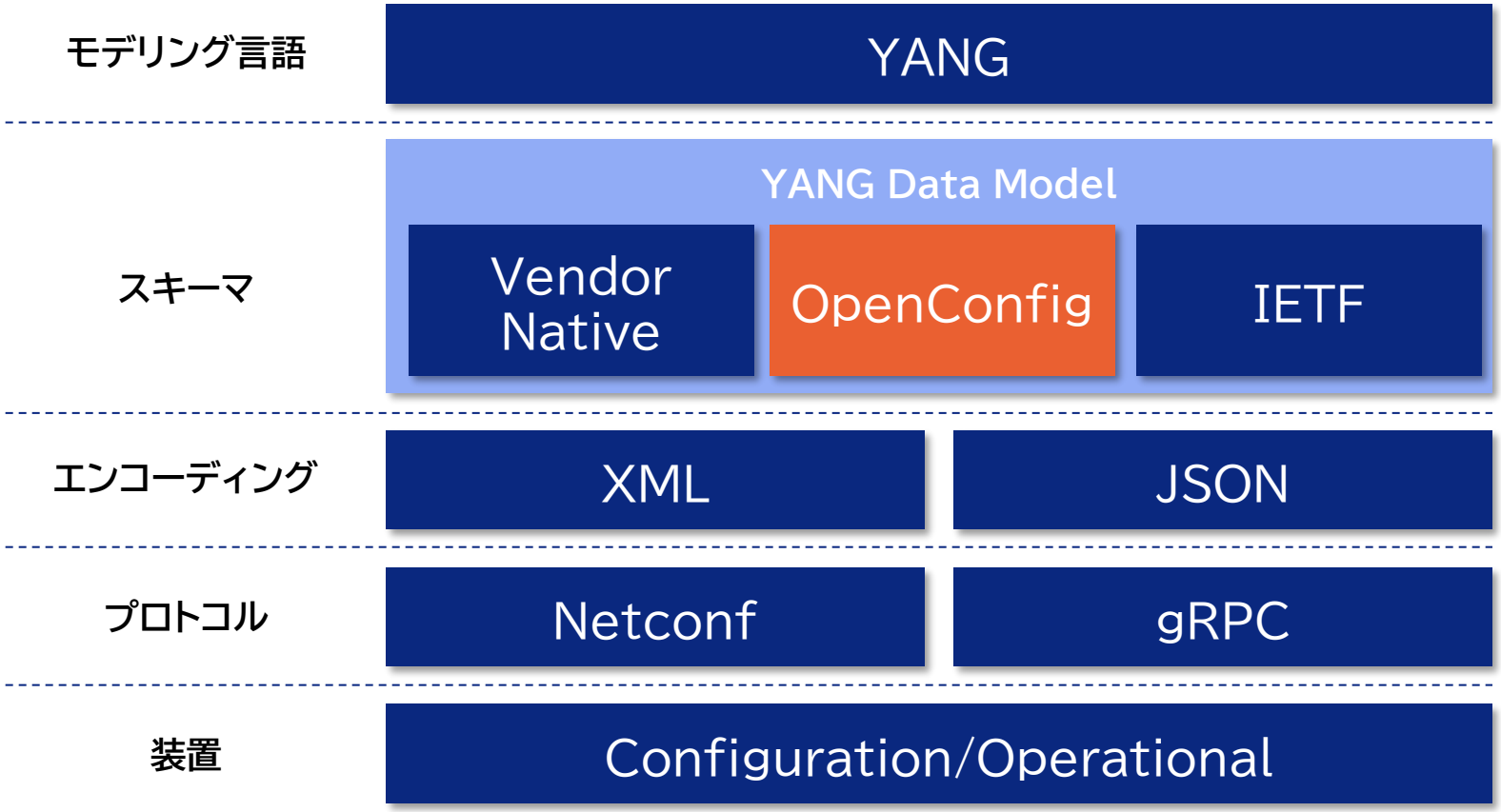
Cisco　Juniper
Nokia　Arista
SONiC　DELL
Ciena

© 2024 KDDI

OpenConfigの位置づけ

OpenConfig：ルータ設定や各種統計情報に関するデータ構造の定義方法の一つ

- 他のレイヤとの依存関係はないため、エンコーディングやプロトコルは環境に応じて選択可能



© 2024 KDDI

OpenConfigのモデル構造例

OpenConfigではモデリングしてない機能もある一方、オペレーターが利用する機能を中心に実装が進んでいる

- 下記は、InterfaceのOpenConfigデータモデル
- YANGで定義されており、各module別にツリー構造となっている

openconfig-interfaces	module	
interfaces	container	
interface[name]	list	
name	leaf	leafref
config	container	
name	leaf	string
type	leaf	identityref
mtu	leaf	uint16
loopback-mode	leaf	oc-opt-types:loopback-mode-type
description	leaf	string
enabled	leaf	boolean
oc-vlan:tpid	leaf	identityref
oc-if-sdn:forwarding-viable	leaf	boolean
oc-p4rt:id	leaf	uint32
state	container	
name	leaf	string
type	leaf	identityref
mtu	leaf	uint16
loopback-mode	leaf	oc-opt-types:loopback-mode-type
description	leaf	string
enabled	leaf	boolean
ifindex	leaf	uint32
admin-status	leaf	enumeration
oper-status	leaf	enumeration
last-change	leaf	oc-types:timeticks64
logical	leaf	boolean
management	leaf	boolean
cpu	leaf	boolean
counters	container	
oc-vlan:tpid	leaf	identityref
oc-port:hardware-port	leaf	leafref
oc-transceiver:transceiver	leaf	leafref
oc-transceiver:physical-channel	leaf-list	leafref

https://openconfig.net/projects/models/schemadocs/jstree/openconfig-interfaces.html#

© 2024 KDDI

OpenConfig WG参加企業

ハイパースケーラーや大手海外キャリアが参加（2024/6月時点:30社）

Google
AT&T
Microsoft
Bloomberg
Netflix
Cloudflare
British Telecom
Facebook
Comcast
Oracle
Tencent
Baidu
Verizon
Level 3
Cox Communications
Alibaba
Telefonica
GoDaddy
Yahoo!
Apple
Jive Communications
Viasat
LinkedIn
Digital Ocean
Deutsche Telekom / Terastream
Bell Canada
SK Telecom
Kuaishou
Sony Interactive Entertainment
Amazon Web Services

https://www.openconfig.net/about/participants/

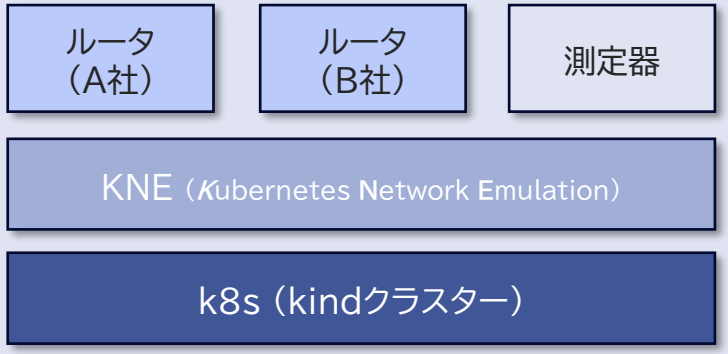
© 2024 KDDI

OpenConfig関連のツール類

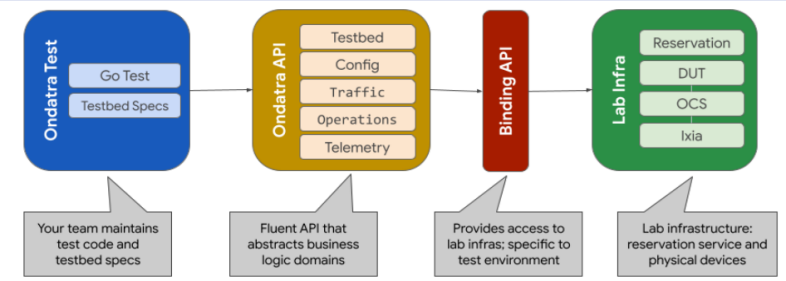
OpenConfig WGでは、データモデルの作成だけでなく、関連ツール類も開発

■ 検証自動化関連では、下記のような関連ツール類が開発・提供されている

NWエミュレーションツール KNE
（*Kubernetes Network Emulation*）

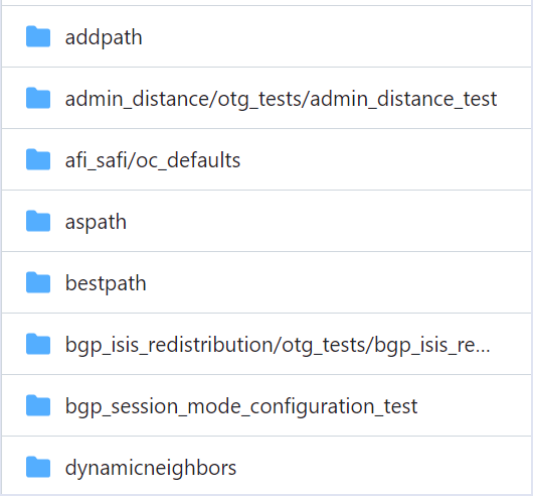


検証自動化フレームワーク ONDATRA
（*Open Network Device Automated Test Runner and API*）



詳細は次頁以降参照

検証自動化サンプルコード featureprofiles



(KNE) https://github.com/openconfig/kne
(ONDATRA) https://github.com/openconfig/ondatra
(featureprofiles) https://github.com/openconfig/featureprofiles

© 2024 KDDI

JANOG54講演

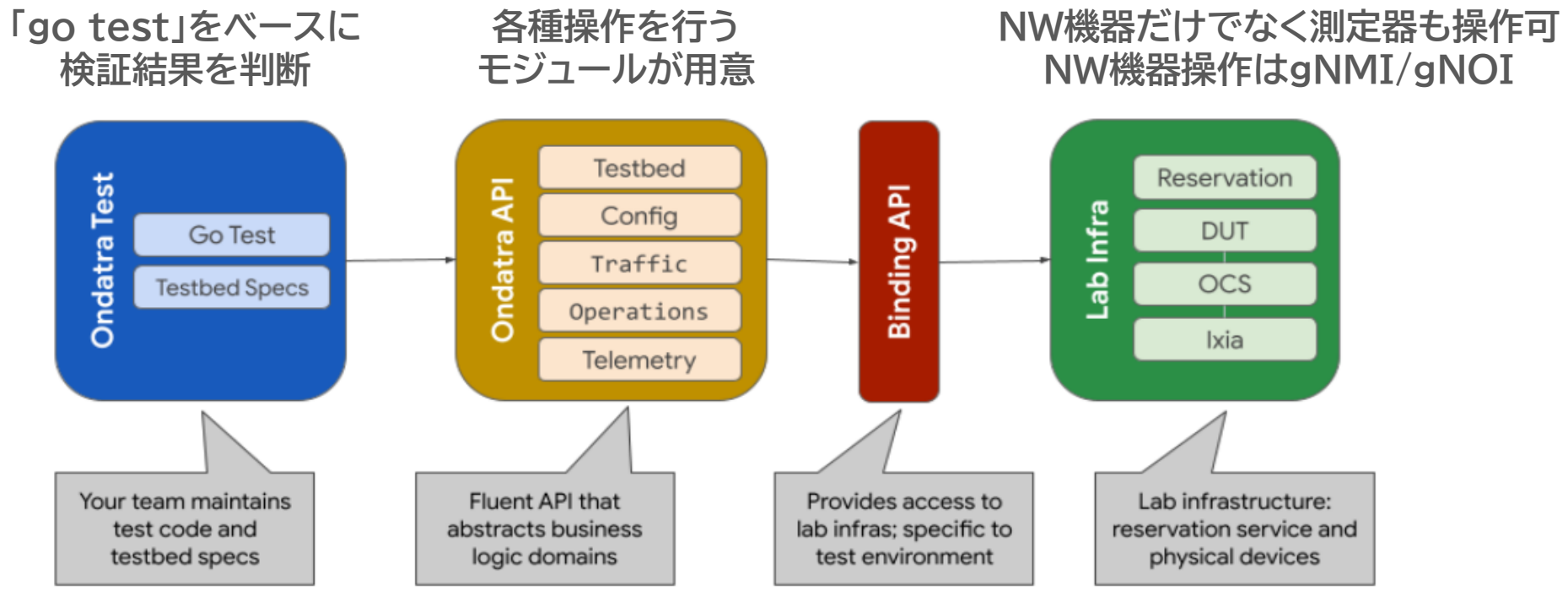
検証自動化フレームワーク

Tomorrow, Together KDDI
おもしろいほうの未来へ。 au

検証自動化フレームワーク概要

ONDATRA（*Open Network Device Automated Test Runner and API*）

- OpenConfigの利用を前提とした検証自動化フレームワーク（Vender Nativeモデルも利用可）
- 主にGoogleが作成・公開しているOSS。Go言語ベースで、go testを利用して自動判定
- NW機器 及び 測定器に対する設定変更/操作/情報取得を容易にする各種モジュールが用意



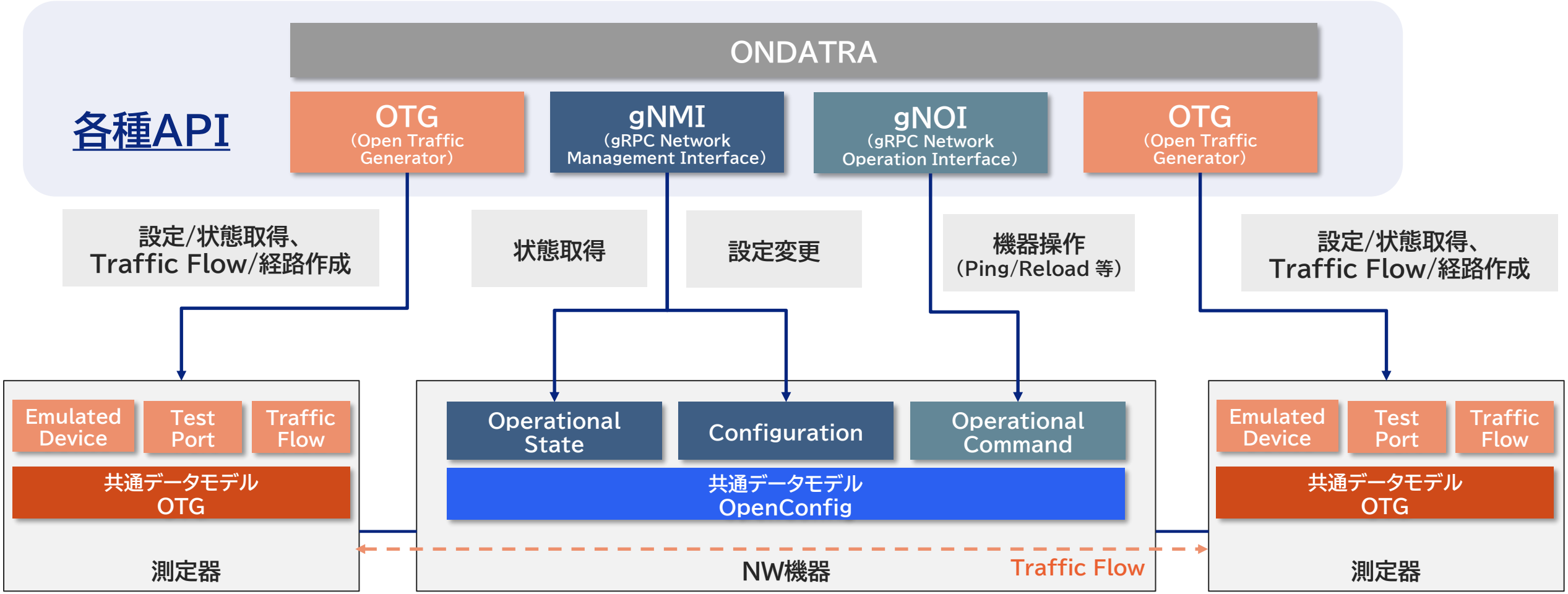
https://docs.google.com/viewer?url=https://raw.githubusercontent.com/openconfig/ondatra/main/internal/tour/tour.pdf

© 2024 KDDI

ONDATRA 各種API

設定変更/操作/情報取得を行う主なAPI：gNMI/gNOI/OTG

■ 他には、CLIコマンドを投入できるRaw APIや、レポート出力をするReport APIなどもある



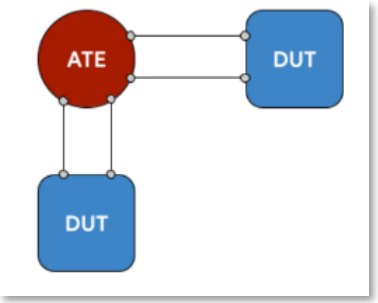
© 2024 KDDI

ONDATRA 各種ファイル

下記2つの定義ファイルの作成により検証項目の自動化が可能

① Testbedファイル

テストトポロジーの定義ファイル
（使用するDUT/ATEと接続性を定義）




```
duts {
  id: "dut"
  vendor: ARISTA
  ports {
    id: "port1"
  }
  ports {
    id: "port2"
  }
}
ates {
  id: "ate"
  vendor: IXIA
  ports {
    id: "port1"
  }
  ports {
    id: "port2"
  }
}
links {
  a: "dut:port1"
  b: "ate:port1"
}
links {
  a: "dut:port2"
  b: "ate:port2"
}
```

② Testシナリオファイル

各試験項目のオペレーションの定義ファイル
（事前設定、試験実施、結果判断など一連の検証工程を定義）

```go
func TestCreateTopology(t *testing.T) {
  ate := ondatra.ATE(t, "ate")
  ap1 := ate.Port(t, "port1")
  ap2 := ate.Port(t, "port2")
  top := ate.Topology().New()
  if1 := top.AddInterface("if1").WithPort(ap1)
  if2 := top.AddInterface("if2").WithPort(ap2)
  if1.IPv4().
    WithAddress("192.168.31.2/30").
    WithDefaultGateway("192.168.31.1")
  if2.IPv4().
    WithAddress("192.168.32.2/30").
    WithDefaultGateway("192.168.32.1")
  net := if2.AddNetwork("net")
  net.IPv4().
    WithAddress("192.168.40.0/30").
    WithCount(100)
  top.Push(t).StartProtocols(t)
}
```

go testファイル(xxxx_test.gp)

- https://docs.google.com/viewer?url=https://raw.githubusercontent.com/openconfig/ondatra/main/internal/tour/tour.pdf
- https://github.com/openconfig/ondatra/blob/main/internal/tour/snippets_testbed.textproto

© 2024 KDDI

技術実証（PoC）

PoCでの評価内容

自動化フレームワーク及び、データモデルの適用度を評価

① 自動化フレームワーク(ONDATRA) ：各検証項目を**どの程度自動化できるか？**

- 今回のPoCでは、16項目が対象

② データモデル(OpenConfig/OTG)：**どの程度機器操作ができるか？**

本PoCでの対象検証項目(一部抜粋)

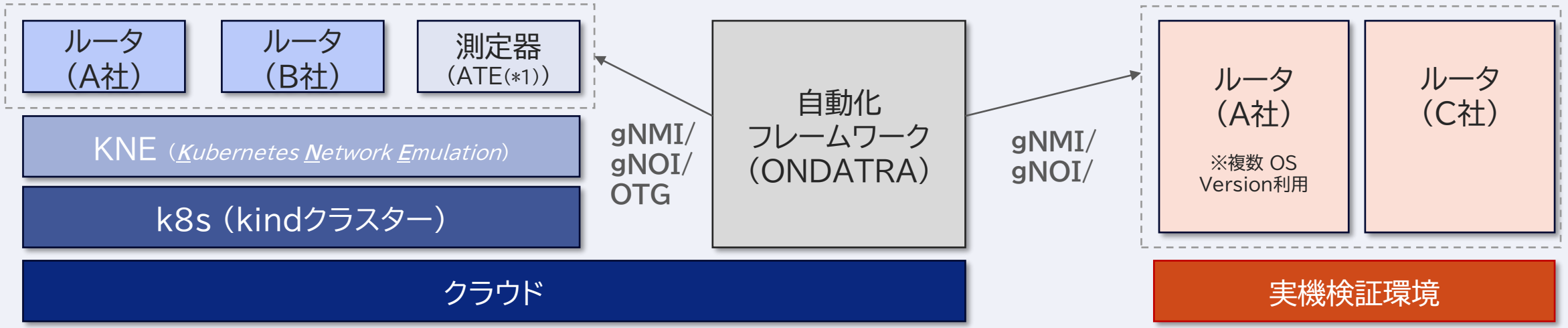
制御対象	検証項目
ルータのみ	I/F shutし、LinkDown・疎通NGであること、及びエラーカウントが無いこと
	1portあたりの最大sub-if設定数が閾値以上であること
ルータ ＋ コンソールサーバ	筐体再起動を行い、正常に対象OSが立ち上がること
ルータ ＋ 測定器	ARPの最大エントリー数が閾値以上であること
	PE-P間で、PE側I/F shudownした際の断時間が基準値以内であること
ルータ ＋ 測定器 ＋ キャプチャ	設定されたMTUサイズを超えたpacket（ DF-bit off ）がフラグメントされること
	iBGP IPv4 Neighbor間のcontrol packetが、優先queueにmappingされること

© 2024 KDDI

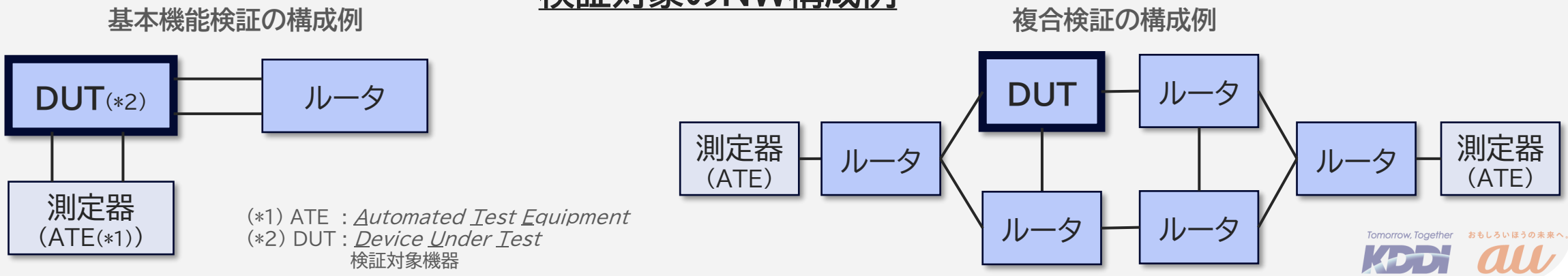
PoC構成

クラウド上に仮想環境を構築し、主に仮想環境でPoCを実施。
一部操作は実機ルータでも実施。

PoC環境構成



検証対象のNW構成例



(*1) ATE : *Automated Test Equipment*
(*2) DUT : *Device Under Test*
検証対象機器

© 2024 KDDI

PoC結果概要

自動化フレームワーク及び、データモデルの適用度を評価

① 自動化フレームワーク(ONDATRA)：各検証項目をどの程度自動化できるか？

- **16項目中、15項目は自動化できる**ことを確認
  (コンソール経由での操作はフレームワーク未対応のため、検証方法の変更が必要)
- Go言語経験者が少なかったが、**コーディングは問題なし**（生成AIに感謝！）

② データモデル(OpenConfig/OTG)：どの程度機器操作ができるか？

- OpenConfig/OTGを用いて、**NW機器/測定器の操作が可能**であることを確認
- ただし、**モデル自体が無い** あるいは、**モデルがあっても機器側が未対応**の操作が存在
  (Vendor Nativeモデルの利用やCLIの併用が必要)

詳細は次頁以降

© 2024 KDDI

PoCで分かったこと

すこし細かい話に・・・

NW機器への設定投入（1/2）

マルチベンダーのNW機器へ、同一のコード（OpenConfigパス）で設定投入が可能

【コード例】I/F shutdown

```
func TestIfShut (t *testing.T) {
    dut := ondatra.DUT(t, routerName)
    port := dut.Port(t, portName)

  gNMI Updateで設定投入
    gnmi.Update(t, dut, gnmi.OC().Interface(port.Name()).Enabled().Config(), false)
}
```

「true」ならno shut

内部で変換される

```
{
  prefix: {
    target: "router1"
  }
  update: {
    path: {
      origin: "openconfig"
      elem: {
        name: "interfaces"
      }
      elem: {
        name: "interface"
        key: {
          key: "name"
          value: "GigabitEthernet0/0/0/0"
        }
      }
    }
    val: {
      json_ietf_val: "false"
    }
  }
}
```

どちらにも設定投入出来た！

CLIで設定を見てみると･･･

#sh run int Gi0/0/0/0

interface GigabitEthernet0/0/0/0
 shutdown
!

＜ A社機器 ＞　＜ B社機器 ＞

CLIで設定を見てみると･･･

#sh run int Ethernet1

interface Ethernet1
  shutdown
!

© 2024 KDDI

NW機器への設定投入（2/2）

マルチベンダーのNW機器へ、同一のコード（OpenConfigパス）で設定投入が可能

【コード例】BGP IPv4 Neighbor設定

```
func TestBgpIpv4Nei(t *testing.T) {
    dut := ondatra.DUT(t, routerName)

    proto := &oc.Root{}
    protoConf := proto.GetOrCreateNetworkInstance("default").GetOrCreateProtocol(oc.PolicyTypes_INSTALL_PROTOCOL_TYPE_BGP, "default")
    bgp := protoConf.GetOrCreateBgp()
    global := bgp.GetOrCreateGlobal()
    global.SetAs(localAS)
    global.SetRouterId(routerId)
    global.GetOrCreateAfiSafi(oc.BgpTypes_AFI_SAFI_TYPE_IPV4_UNICAST).SetEnabled(true)
    nv4 := bgp.GetOrCreateNeighbor(neighborIP)
    nv4.SetPeerAs(peerAS)
    nv4.GetOrCreateAfiSafi(oc.BgpTypes_AFI_SAFI_TYPE_IPV4_UNICAST).SetEnabled(true)

    gnmi.Update(t, dut, gnmi.OC().NetworkInstance("default").Protocol(oc.PolicyTypes_INSTALL_PROTOCOL_TYPE_BGP, "default").Config(), protoConf)
}
```

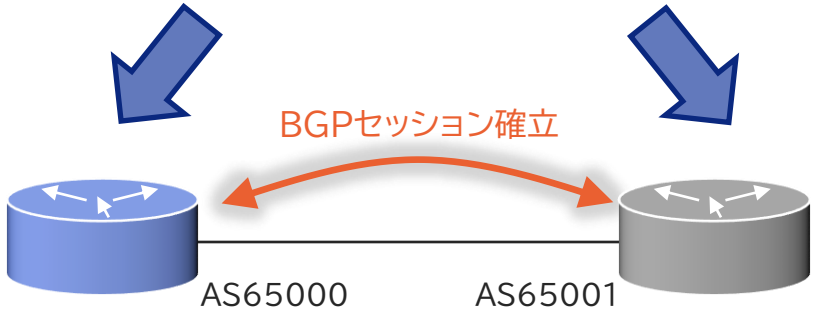
各種パラメータ設定

gNMI Updateで設定投入

↑で作成した設定

```
#sh run router bgp
router bgp 65000
 bgp router-id 192.0.2.1
 address-family ipv4 unicast
 !
 neighbor 192.0.2.2
  remote-as 65001
  address-family ipv4 unicast
  !
 !
!
```

どちらにも設定投入出来た！

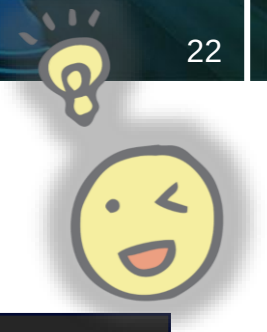


＜ A社機器 ＞　＜ B社機器 ＞

```
#sh run section router bgp
router bgp 65001
  router-id 192.0.2.2
  bgp default ipv4-unicast transport ipv6
  neighbor 192.0.2.1 remote-as 65000
  !
  address-family ipv4
    neighbor 192.0.2.1 activate
!
```

© 2024 KDDI

NW機器からの情報取得



マルチベンダーのNW機器から、同一のコード（OpenConfigパス）で情報取得が可能

【コード例】BGP IPv4 Neighbor セッション状態確認

gNMI Getでセッション状態を取得

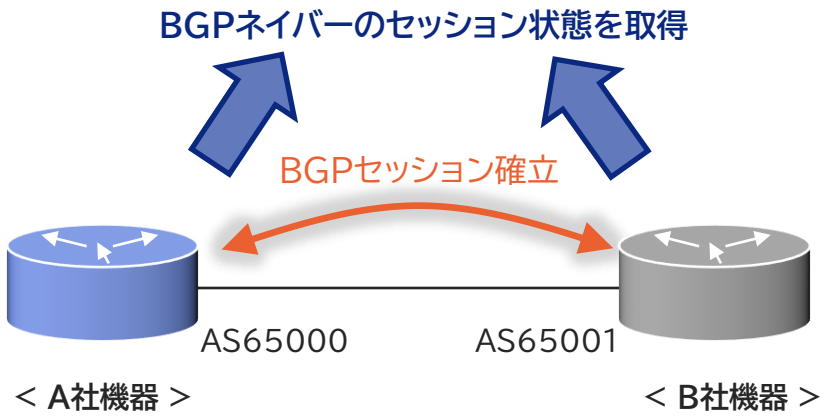
```
func TestBgpNbrSessionChk(t *testing.T) {
    dut := ondatra.DUT(t, routerName)

    bgpNbrSessionStatus := gnmi.Get(t, dut,
    gnmi.OC().NetworkInstance(niName).Protocol(oc.PolicyTypes_INSTALL_PROTOCOL_TYPE_BGP,instName).Bgp().Neighbor(neighborIP).SessionState().State())

    if bgpNbrSessionStatus == oc.Bgp_Neighbor_SessionState_ESTABLISHED {
        t.Log("Chk OK : The Bgp Neighbor Session State is ESTABLISHED.")
    } else {
        t.Error("Chk NG : The Bgp Neighbor Session State is not ESTABLISHED.")
    }
}
```

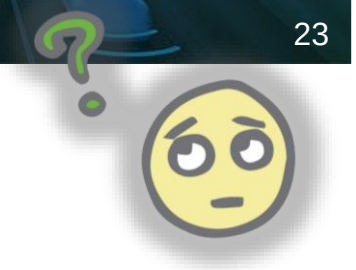
セッション状態が”ESTABLISH”かどうか判定

どちらも情報取得が出来た！ & パースも不要！



© 2024 KDDI

異VersionのNW機器への設定投入



OS Versionによって、OpenConfigのパス指定が微妙に異なる
⇒ 複数のOS Versionへの利用時は注意が必要

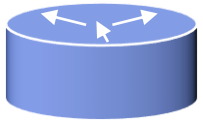
【コード例】I/F shutdown

```
func TestIfShut (t *testing.T) {
    dut := ondatra.DUT(t, routerName)
    port := dut.Port(t, portName)
    gnmi.Update(t, dut, gnmi.OC().Interface(port.Name()).Enabled().Config(), false)
}
```

```
{
update: {
    path: {
      origin: "openconfig"
      elem: {
        name: "interfaces"
      }
      elem: {
        name: "interface"
        key: {
          key: "name"
          value: "GigabitEthernet0/0/0/0"
        }
      }
    }
    val: {
      json_ietf_val: "false"
    }
  }
}
```

投入OK

Version x.x.x（新しめ）



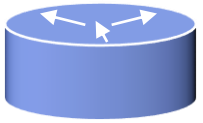
```
{
update: {
    path: {
      origin: "openconfig"
      elem: {
        name: "interfaces"
      }
      elem: {
        name: "interface"
        key: {
          key: "name"
          value: "GigabitEthernet0/0/0/0"
        }
      }
    }
    val: {
      json_ietf_val: "false"
    }
  }
}
```

"origin"フィールドにて、使用するスキーマを指定
この指定では機器側が認識出来ない（なぜ!?）

origin指定無し or
origin:"openconfig-interfaces" の場合は問題なし
⇒ origin指定を省略する形でコード改修

投入NG

Version y.y.y（古め）



【エラー内容】
gNMI: set-request : : Module 'openconfig' not recognized or supported for the specified operation

© 2024 KDDI

【参考】gNMIにおけるoriginの指定について

デフォルトのoriginは”openconfig”

- “origin”にて、使用するスキーマ(OpenConfig、Vendor Native)を指定

Mixing Schemas in gNMI

Contributors: aashaikh, hines, robjs, csl
October 2016, Updated May 2018
Implementation Status: Merged

Problem Statement

Today's network devices support multiple schema representations of their underlying configuration and telemetry databases. Particularly, devices typically support:

- A "native" schema. This representation generally 1:1 maps to the underlying schema of the database storing configuration and telemetry variables on the device.
- A vendor-neutral schema. This representation transforms the underlying schema into a vendor-neutral model, namely OpenConfig.
- A CLI schema. This represents the text CLI format that has been used to configure the device by humans.

Special Values of `origin`

Origin values are agreed out of band to the gNMI protocol. Currently, special values are registered within this section of this document. Should additional origins be defined, a registry will be defined.

Where the `origin` field is unspecified, its value should default to `openconfig`. It is RECOMMENDED that the origin is explictly set.

デフォルトのorigin
＝“openconfig”

Where the `origin` field is specified as `cli`, the path should be ignored and the value specified within the `Update` considered as CLI input.

https://www.openconfig.net/docs/gnmi/mixed-schema/

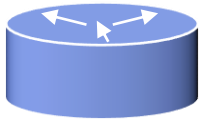
© 2024 KDDI

NW機器側が対応するOpenConfig Version

NW機器側のOS Versionによって、対応するOpenConfigのVersionが異なる

- OpenConfigのVersionだけアップデートすることは出来ない
- 複数のOS Versionへの利用時は**注意が必要（特に古いOS Version）**

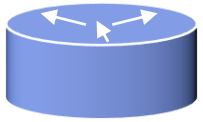
（例）openconfig-network-instance及び、openconfig-interfaces



Version x.x.x（新しめ）

```
$ gnmic -a xxx.xxx.xxx.xxx:nnn -u xxx -p yyy --skip-verify capabilities ¥¥
 | egrep "openconfig-network-instance|openconfig-interfaces"

  - openconfig-network-instance-l3, OpenConfig working group, 0.11.1
  - openconfig-network-instance, OpenConfig working group, 0.15.0
  - openconfig-network-instance-l2, OpenConfig working group, 0.15.0
  - openconfig-network-instance-types, OpenConfig working group, 0.9.3
  - openconfig-interfaces, OpenConfig working group, 2.5.0
```



Version y.y.y（やや古め）

```
$ gnmic -a xxx.xxx.xxx.xxx:nnn -u xxx -p yyy -skip-verify capabilities ¥¥
 | egrep “openconfig-network-instance|openconfig-interfaces”

  - openconfig-network-instance-l3, OpenConfig working group, 0.11.1
  - openconfig-network-instance, OpenConfig working group, 0.15.0
  - openconfig-network-instance-l2, OpenConfig working group, 0.15.0
  - openconfig-network-instance-types, OpenConfig working group, 0.9.1
  - openconfig-interfaces, OpenConfig working group, 2.4.3
```

Vupにより、
一部モデルが追加

public / release / models / interfaces / openconfig-interfaces.yang

Code　Blame　1336 lines (1127 loc) · 40.7 KB ·

```
99      revision "2021-04-06" {
100       description
101         "Add leaves for management and cpu interfaces";
102       reference "2.5.0";
103     }
```

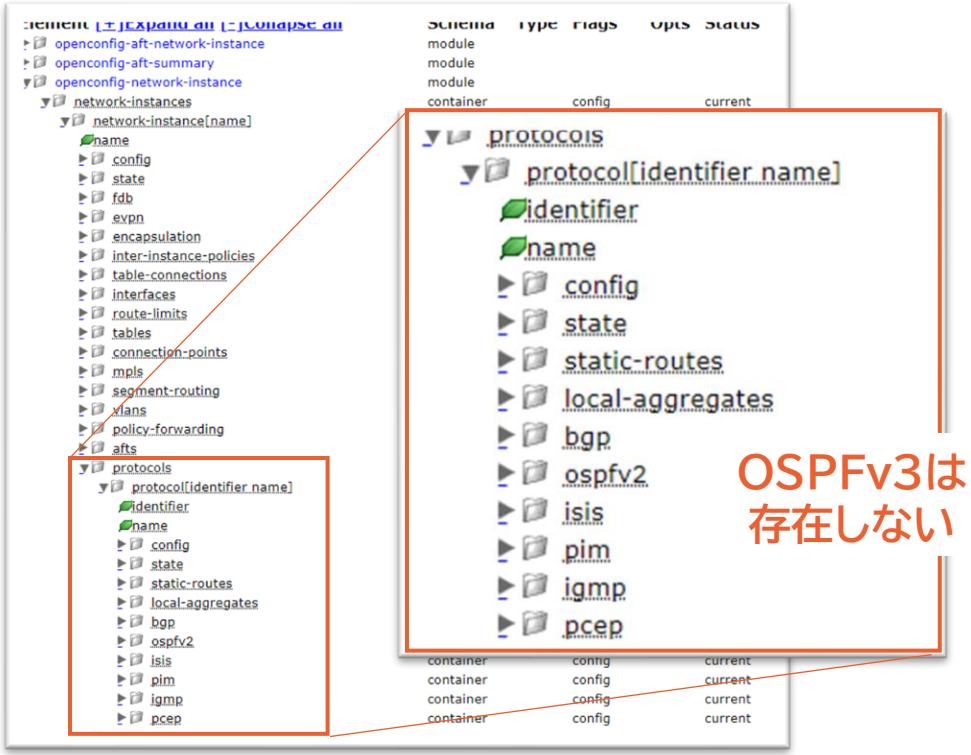
https://github.com/openconfig/public/blob/master/release/models/interfaces/openconfig-interfaces.yang

© 2024 KDDI

OpenConfigモデル未対応の例

OpenConfigで**未対応な操作がある**ため、全てを満たすことは難しい
⇒ CLI操作や、Vendor Nativeモデルの利用も必要

ネットワークプロトコル関連
（openconfig-network-instance）



https://openconfig.net/projects/models/schemadocs/jstree/openconfig-network-instance.html

routing-policy関連
（openconfig-routing-policy）

Routing-policy内の
Action部分で、
IOS-XRの「pass」に
相当するものが無い

policy-result(Action)の対応付け

OpenConfig	REJECT_ROUTE	ACCEPT_ROUTE	（該当なし）
Cisco IOS-XR(CLI)	drop	done	pass

Cisco IOS-XR(CLI)

```
route-policy XXXX
   {drop|done|pass}
end-policy
```

OpenConfigでは
「pass」を設定できない

© 2024 KDDI

NW機器側のOpenConfig未対応の例

OpenConfigモデルがあっても、NW機器側未対応の項目もある

- 未対応のパスは、deviationsファイルに記載されている

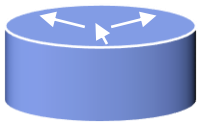
ベンダ/OS別のOpenConfigモデル未対応の確認方法(一例)

```
$ git clone https://github.com/YangModels/yang.git; cd yang

$ grep -B 1 not-supported vendor/ <ディレクトリパス(省略)> /xxxxxx-openconfig-xxxxxxxxxx-deviations.yang
~省略~

<表示例>
 deviation /oc-netinst:network-instances/oc-netinst:network-instance/oc-netinst:protocols/oc-netinst:protocol/oc-netinst:bgp/oc-netinst:global/oc-netinst:afi-safis/oc-netinst:afi-safi/oc-netinst:graceful-restart/oc-netinst:config/oc-netinst:enabled {
    deviate not-supported;
```

某OS Versionの場合



BGP関連の未対応パス（一部抜粋）

- graceful-restart
- use-multiple-paths
- always-compare-med
- max-prefixes
- send-community

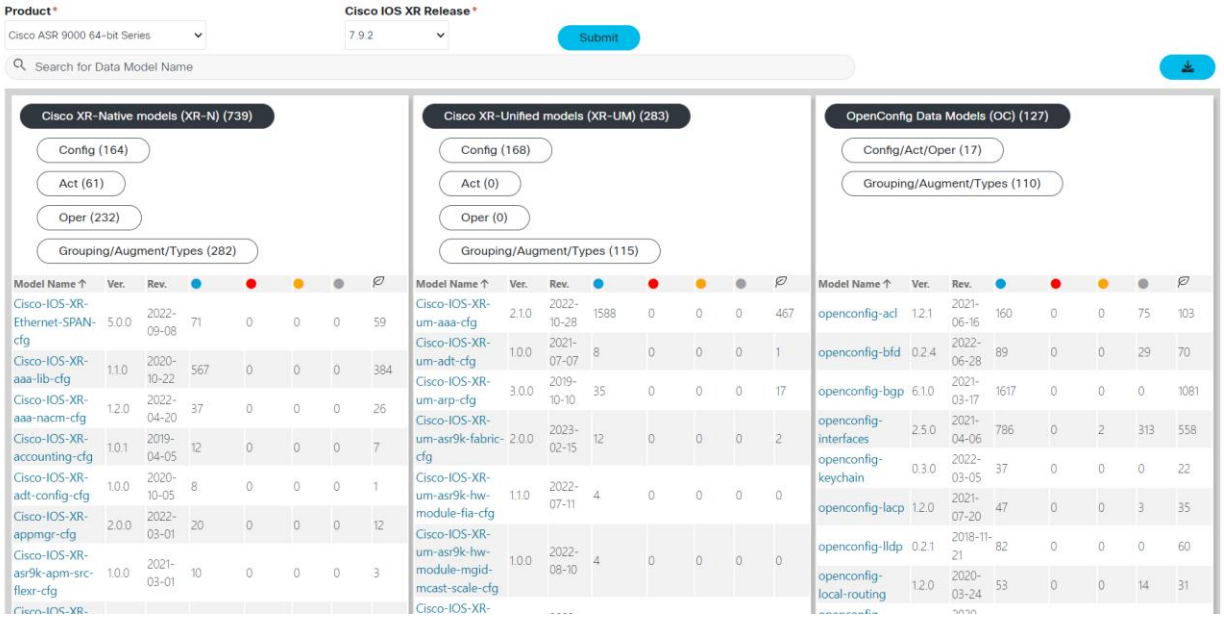
© 2024 KDDI

【参考】お役立ち情報：各ベンダで対応しているYANGモデル情報の確認方法

Cisco
Feature Navigator

- 機器種別/OS Ver別に、**利用しているYANGモデルVersion**が確認可能

(例) ASR9k, IOS-XR 7.9.2の場合



https://cfnng.cisco.com/ios-xr/yang-explorer/view-data-model

Juniper
OpenConfig User Guide

- 各OS Ver別に、**利用しているOpenConfigのVersion**が確認可能
- **OpenConfigパスに対応するJunos Configの**確認が可能

マッピング情報

Table 41: IFD Configuration

Command Name	*OpenConfig Command Path*	*Junos Configuration*
Interface Type	/interfaces/*interface*/config/type	Not supported. Type is derived from the interface name.
Interface MTU	/interfaces/*interface*/config/mtu	set interface *interface* mtu
Interface Name	/interfaces/*interface*/config/name	set interface *interface*
Interface Description	/interfaces/*interface*/config/description	set interface *interface* description
Interface Enabled/ Disabled	/interfaces/*interface*/config/enabled	set interface *interface* disabled set interface *interface* enabled

https://www.juniper.net/documentation/us/en/software/junos/open-config/open-config.pdf

© 2024 KDDI

測定器への設定投入

測定器に対しても、OTG APIを利用して設定投入が可能

【コード例】BGP IPv4 Neighbor及び、経路負荷設定

```
func TestATEBgpConfig(t *testing.T)  {
    ate := ondatra.ATE(t, a.Name)
    gosnappiConfig := gosnappi.NewConfig()
    port := gosnappiConfig.Ports().Add().SetName(a.Port1.Name)
    dev := gosnappiConfig.Devices().Add().SetName(a.Port1.Name+"Dev")
    eth := dev.Ethernets().Add().SetName(a.Port1.Eth).SetMac(a.Port1.MAC)
    eth.Connection().SetPortName(port.Name())
    ipv4 := eth.Ipv4Addresses().Add().SetName(a.Port1.Name+"IPv4")
    ipv4.SetAddress(a.Port1.IPv4Address).SetGateway(r1.Port1.IPv4Address).SetPrefix(a.Port1.IPv4PrefixLength)

    bgp := dev.Bgp().SetRouterId(a.Port1.IPv4Address)
    bgp4Peer := bgp.Ipv4Interfaces().Add().SetIpv4Name(ipv4.Name()).Peers().Add().SetName(a.Port1.Name+"BGP4Peer"))
    bgp4Peer.SetPeerAddress(ipv4.Gateway()).SetAsNumber(a.BgpNei.localAS).SetAsType(gosnappi.BgpV4PeerAsType.EBGP)
    bgp4Peer.Capability().SetIpv4UnicastAddPath(true).SetIpv6UnicastAddPath(false)

    routes4 := bgp4Peer.V4Routes().Add().SetName(bgp4Peer.Name() + "-v4-routes")
    routes4.SetNextHopIpv4Address(ipv4.Address()).
            SetNextHopAddressType(gosnappi.BgpV4RouteRangeNextHopAddressType.IPV4).
            SetNextHopMode(gosnappi.BgpV4RouteRangeNextHopMode.MANUAL)
    routes4.Addresses().Add().SetAddress("172.16.0.0").SetPrefix(24).SetCount(1000)

    ate.OTG().PushConfig(t, gosnappiConfig)
    ate.OTG().StartProtocols(t)
}
```

- I/F設定
- BGPピア設定
- BGP経路負荷設定
  左記のコードでは1,000経路
  (172.16.0.0/24～172.19.231.0/24)
- 設定投入

BGP Nei、経路負荷の設定投入が出来た！

© 2024 KDDI

OTGモデル対応状況

OTGを利用して、仮想デバイスやトラフィックフロー作成が可能
ただし、OpenConfig同様に、モデル未対応の項目もあり

（未対応の例：OSPF、PIM、BGP Labeled Unicast）

OTGモデルの対応内容

Features

OTG is an actively developed specification, with contributions from real use cases. The model defines the following components of a traffic generator configuration:

- **Test Ports** with Layer 1&2 capabilities, including:
  - LLDP, LAG, LACP
- **Emulated Devices** with Layer 2&3 features:
  - IPv4, IPv6 interfaces
  - ARP, IPv6 ND
  - BGP, IS-IS routing protocols
- **Traffic Flows**
  - associated with either Test Ports, or Emulated Devices
  - expressing L2-4 properties like Ethernet, IPv4/IPv6, TCP/UDP
  - stateless or stateful capabilities for transport protocols
  - with implementation-specific application payload
- Run-time **Metrics** and traffic **Capture** capabilities

OSPFv2/3やPIMは未対応

https://otg.dev/

© 2024 KDDI

Tomorrow, Together KDDI おもしろいほうの未来へ。au

PoCで分かったことのまとめ

検証自動化に対してOpenConfigの活用がある程度有効であるが、
モデル未対応などの課題がある。

- OpenConfigを利用し、マルチベンダーのNW機器に対して設定投入や状態確認が可能
- OTGを利用し、測定器に対して設定投入や状態確認、パケットキャプチャが可能

- OpenConfig、OTGともに、モデル未対応の項目がある
  - OpenConfig：OSPFv3 等
  - OTG：OSPFv2/v3、PIM 等
  - ⇒ モデル拡充や、Native YANGの併用が必要
- NW機器側のOS Versionによって、OpenConfigのパス指定の内容が異なる（例：パスのorigin指定）

© 2024 KDDI

今後の課題

今後の課題

実検証への利用を進めつつ、OpenConfigのモデル拡充を図る

社内

① 検証業務の一部への利用開始

- 実検証の一部への利用開始、更なる知見の深化

② CI/CDの実装

- OSやモデルのVupに追従するためのCI/CD

③ 既存の検証自動化ツールとの棲み分け整理

- CLIベースの既存自動化ツールとの棲み分け/使い分けの整理

社外

④ ベンダとの連携（機能実装確認・開発要求）

- ベンダ実装方針のヒアリング
- 機能実装要求・議論

⑤ OpenConfigコミュニティ参加

検討中

- 共通データモデルの実装要求
- ベンダ実装要求の協力

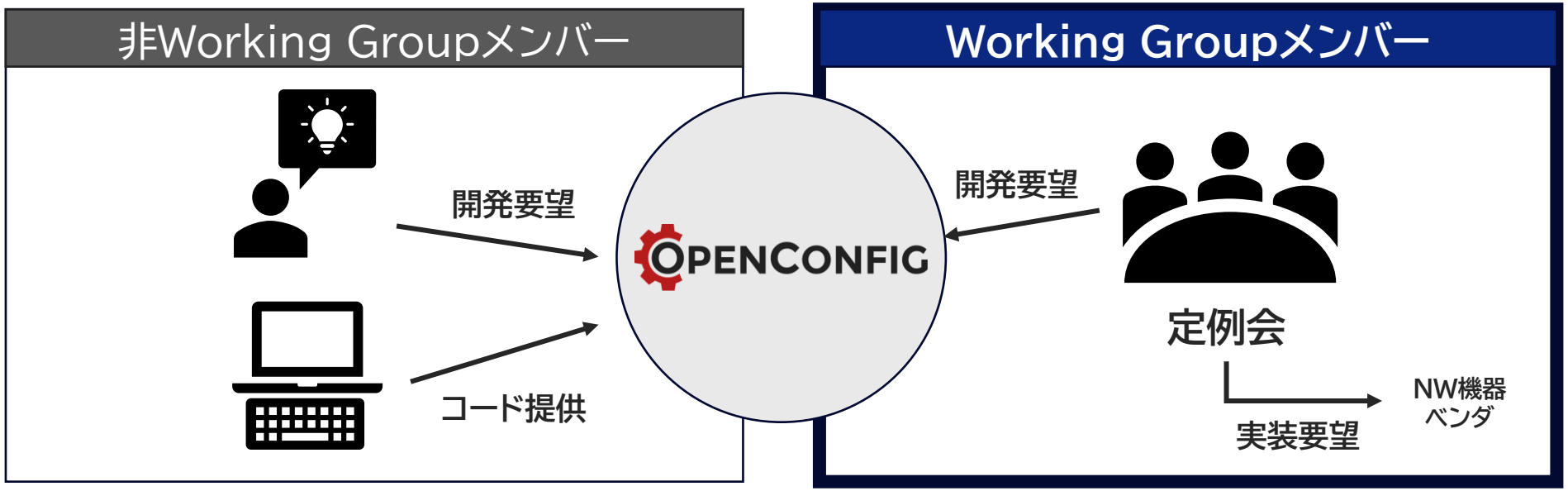
© 2024 KDDI

OpenConfig Project Working Group参加

検討中
仲間募集中！

OpenConfigのモデル拡充を目的に、WGへの参加を検討中

- WG参加によるメリット
  - WGメンバーによる定例会があり、コミュニティ内部からデータモデル拡張要望が可能
  - WG参加企業と連携してベンダ実装要求を行うことで、機器実装面の加速・要望反映が期待



「対応してない項目があるから使えない…」ではなく、
我々自ら「使えるものにしていく！」
（同志の方々、ぜひ一緒にやりましょう！）

© 2024 KDDI

まとめ・議論したいポイント

まとめ

モチベーション	NW検証業務を**効率化**したい
アプローチ	**自動化** ＋ **OpenConfig**の活用
PoCを通して分かったこと	• マルチベンダー機器の検証自動化に対して、**OpenConfigの利用は有効** その一方で、**モデルなし** または、**NW機器側未対応**の操作が存在 • ONDATRAを利用することで、**OpenConfigの利用は容易**
今後	モデル拡充・NW機器側の対応拡充へ向けた取り組み・働きかけ ⇒ ***Model-Driven*** へ変革していきたい （OpenConfigだけでなく、Vender Nativeも利用）

© 2024 KDDI

議論したいポイント

ベンダーニュートラルな自動化やYANGモデルの活用方法に関して、
下記の内容について議論したい

NW機器ベンダー向け

① OpenConfig対応の**モチベーション・スタンス**は？（積極的？ 消極的？）

② OpenConfig WGと**どのように連携**していますか？

オペレーター向け

① **マルチベンダー機器に対する自動化**は、どのように対応してますか？

② **YANGモデル**(OpenConfig/Vendor Native)つて使っていますか？
使っていれば、**どのような課題**がありますか？
それとも、**CLIベースの自動化**が多いですか？

© 2024 KDDI

Tomorrow, Together
KDDI

「つなぐチカラ」を進化させ、
誰もが思いを実現できる社会をつくる。

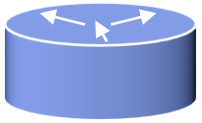
KDDI VISION 2030

参考資料

【参考】ちなみに、gNOIも・・・

gNOI(*gRPC Network Operations Interface*)でのオペレーション操作についても、
NW機器側未対応の項目がある

某OS Versionの場合



全て対応

- **System**
  - Reboot, Ping など
- **File**
  - Put, Get など
- **BGP**
  - ClearBGPNeighbor
- **Diagnostic**
  - Start/Stop BERT(*), Get BERT Result

(*)BERT:Bit Error Rate Test

一部対応

- **Layer2**
  - ClearLLDPInterface のみ
- **Certificate Management**
  - Install, Get など（Load未対応）

未対応

- **MPLS**
- **OS**
- **BootConfig**
- **Debug**
- **Packet Capture** など

© 2024 KDDI