沖縄オープンラボラトリ
Model Driven Network DevOps (MDDO) Projectの紹介

ENOG 74 2022/6/10

ビッグローブ株式会社
基盤本部
ネットワーク技術部

滝口敏行
川口永一郎

自己紹介

・滝口　敏行
所属：ビッグローブ株式会社 基盤本部 ネットワーク技術部
経歴：2018年中途入社 社会人10年目 現在3社目
やっていること：Prometheusを使った運用監視システムの開発、保守。
MAPE・DNSインフラの運用改善、NW自動化の施策検討/推進など

・川口　永一郎
所属：ビッグローブ株式会社 基盤本部 ネットワーク技術部
経歴：2020年新卒 3年目
やっていること：DNS、IPv4 over IPv6基盤の運用

本Projectについて

沖縄オープンラボラトリ※のProjectの一つで、
ネットワーク(NW)運用の高度化を実現するためにNWのトポロジに着目して
NW設計～運用を自動化する仕組みを作る試みを行っているProject

※沖縄オープンラボラトリ(https://www.okinawaopenlabs.org/about-ool)とは、
沖縄を活動基盤とした企業、団体と国や業界、組織の枠を超えて次世代ICT基盤技術の実用化、
普及を推進するオープンに活動する団体

Project参加企業
・TIS株式会社
・ビッグローブ株式会社
・NTTコミュニケーションズ

背景・課題感

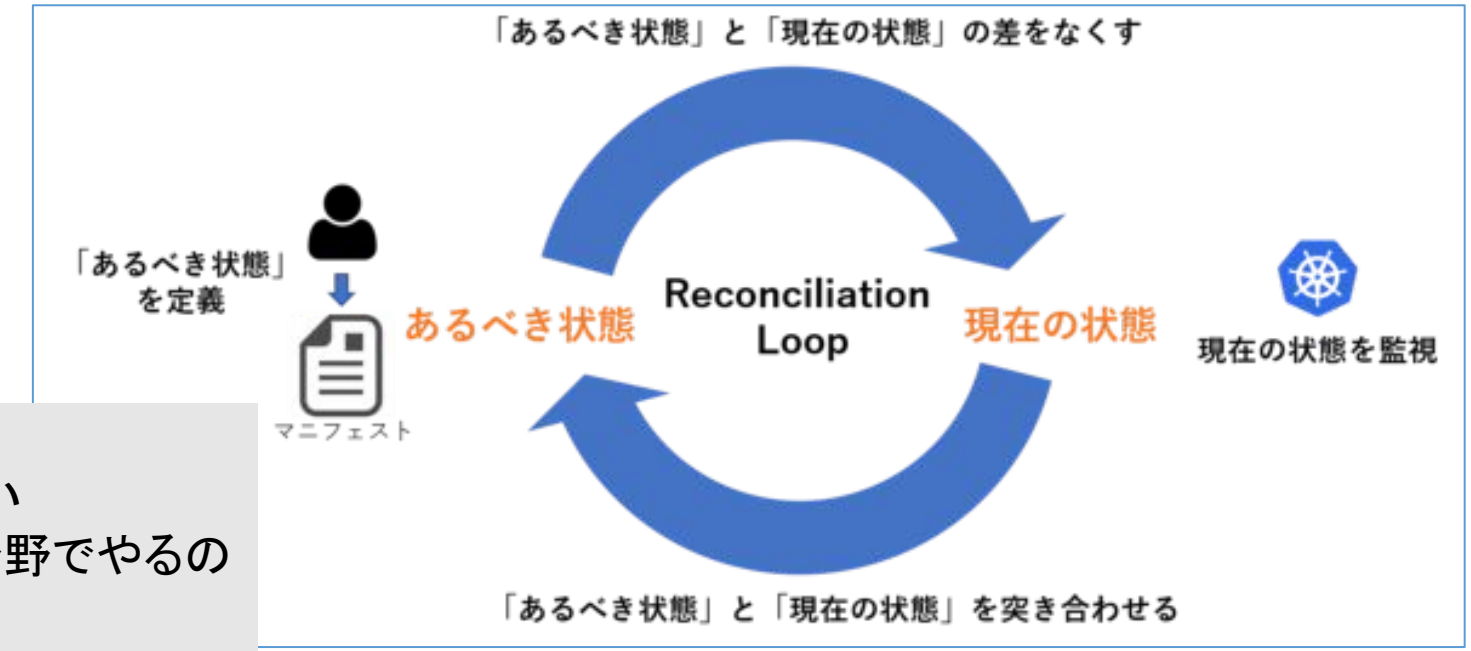
- コンフィグと台帳ベースの人力運用でNWの複雑化に対処するのは限界
  - ノード単体に対する操作の自動化は一般的になったが、ネットワークでは複数ノード間の関係性が見えないと正しい操作はできない
  - ノード間の関係性を把握した自動化への拡張が必要
- 大きなソリューションを入れて従来のやり方を大幅に変えるのは難しく、抵抗感も大きい
  - 市販のNW管理製品等で、構成情報・ノード間接続情報などを基に管理・分析できるものもあるが、ある程度規模のあるシステムでないと合わない
  - 汎用的な製品 ⇔ システム固有の設計・制約・課題：ギャップをどう埋めるか?
  - 既存のしくみを補完し、課題に合せて小さく始めて積み上げるアプローチがほしい

- NW自動化は作業時間の短縮だけでなく
理解・判断の領域へのステップアップ⇒単体から全体の関係性へ
- 宣言的な手法/Reconciliation Loop などのサーバサイドのCI/CDノウハウ・考え方を取り込めないか

[参考]宣言的手法とReconciliation Loop

Kubernetesの突合せ(Reconciliation)ループ

- 理想状態の定義：宣言
- 現在状態との比較
- 理想状態への移行
  → 常に理想状態を維持



既存のNWインフラ環境はきれいにひとつの
プラットフォームとして統合できるわけじゃない
ので、こういうオペレーションをネットワーク分野でやるのはすごく難しい。
しかし、**基準点(理想的な構成)を定義して、
そことのずれを出す、みたいなことができるだけでも効果的なユースケースはあると考えた。**

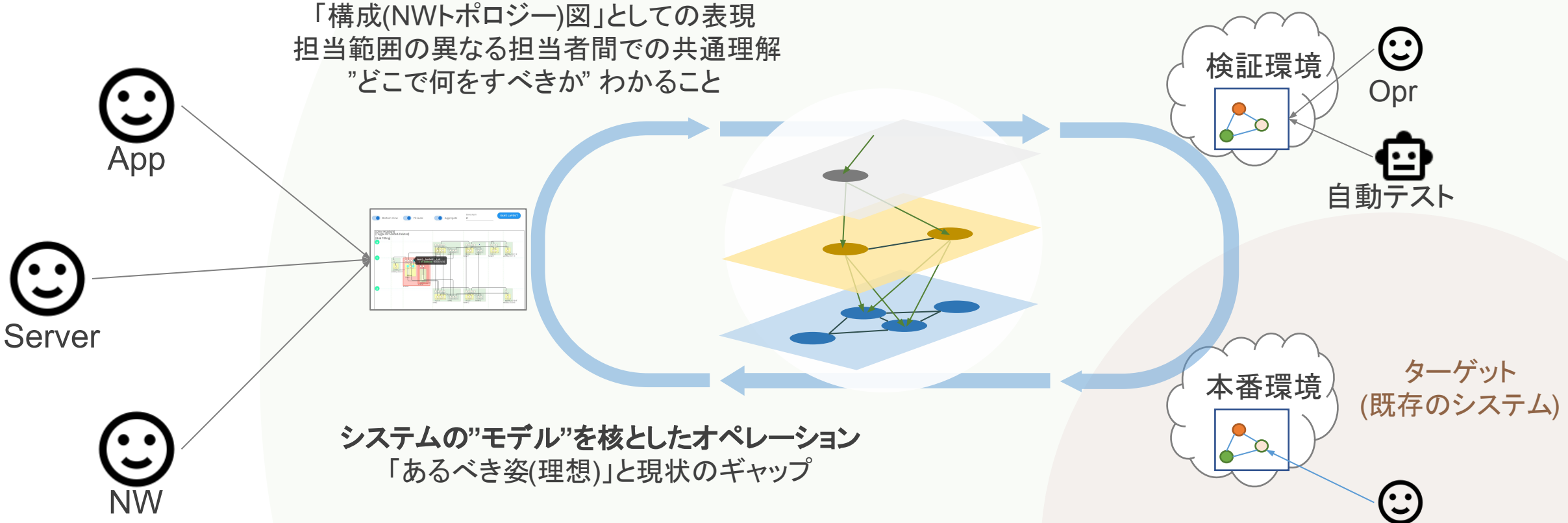
データ永続化と構築運用の自動化を実現する「PostgreSQL on Kubernetes」の仕組み：クラウドネイティブ時代のデータベース（2） - @IT
https://www.atmarkit.co.jp/ait/articles/2105/20/news006.html

実現したいこと

既存システムにWrapする形で
計算機による技術支援を拡充していく

デプロイ前のテストと予測可能性
静的な検査・動的なテスト
練習・トレーニング・育成

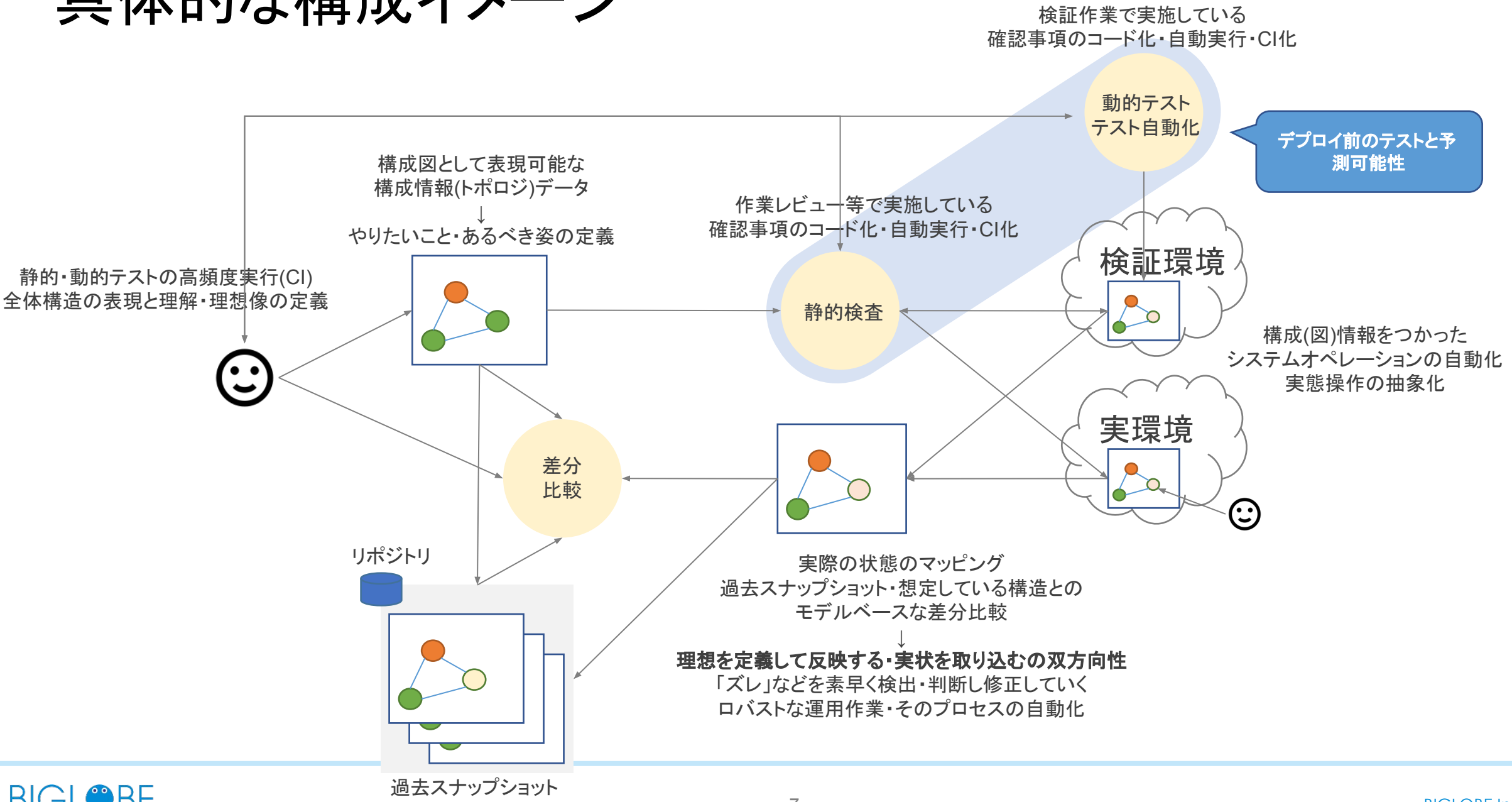
複雑なシステム全体像の俯瞰:
「構成(NWトポロジー)図」としての表現
担当範囲の異なる担当者間での共通理解
”どこで何をすべきが” わかること



システムの”モデル”を核としたオペレーション
「あるべき姿(理想)」と現状のギャップ

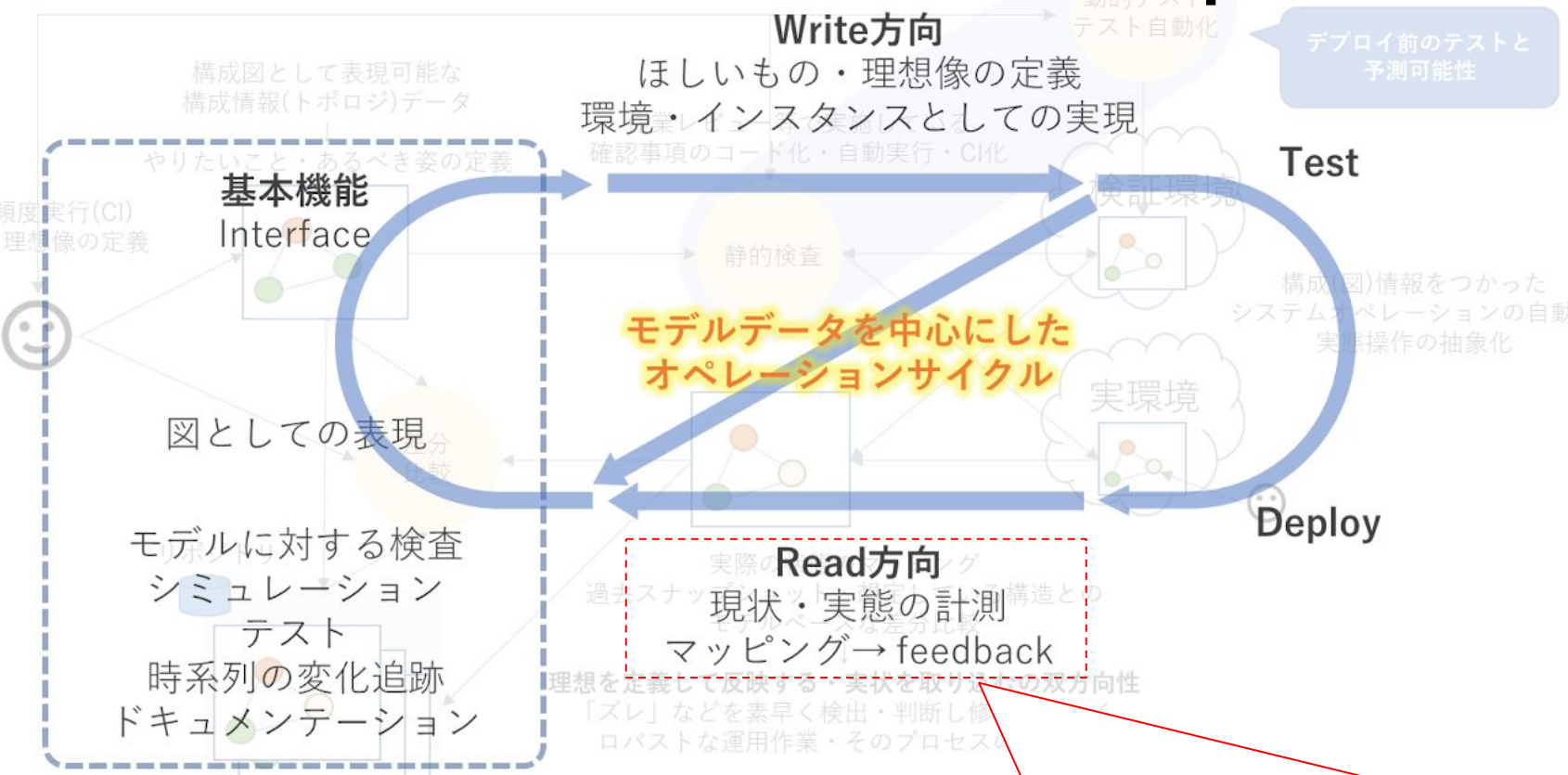
宣言的なシステム運用
自動テストなどソフトウェア開発・クラウド開発の
ベストプラクティスの取り込み

具体的な構成イメージ


検証作業で実施している
確認事項のコード化・自動実行・CI化
動的テスト
テスト自動化
デプロイ前のテストと予測可能性
構成図として表現可能な
構成情報(トポロジ)データ
↓
やりたいこと・あるべき姿の定義
作業レビュー等で実施している
確認事項のコード化・自動実行・CI化
静的・動的テストの高頻度実行(CI)
全体構造の表現と理解・理想像の定義
静的検査
検証環境
構成(図)情報をつかった
システムオペレーションの自動化
実態操作の抽象化
実環境
差分
比較
リポジトリ
実際の状態のマッピング
過去スナップショット・想定している構造との
モデルベースな差分比較
↓
理想を定義して反映する・実状を取り込むの双方向性
「ズレ」などを素早く検出・判断し修正していく
ロバストな運用作業・そのプロセスの自動化
過去スナップショット

今回実装した範囲

- **既存システム・その運用の中にreconciliation loopを取り込む**



今回(2022年5月まで)実装した範囲
・コンフィグからモデルデータを抽出し、
静的検査および経路シミュレーションを自動で行う。

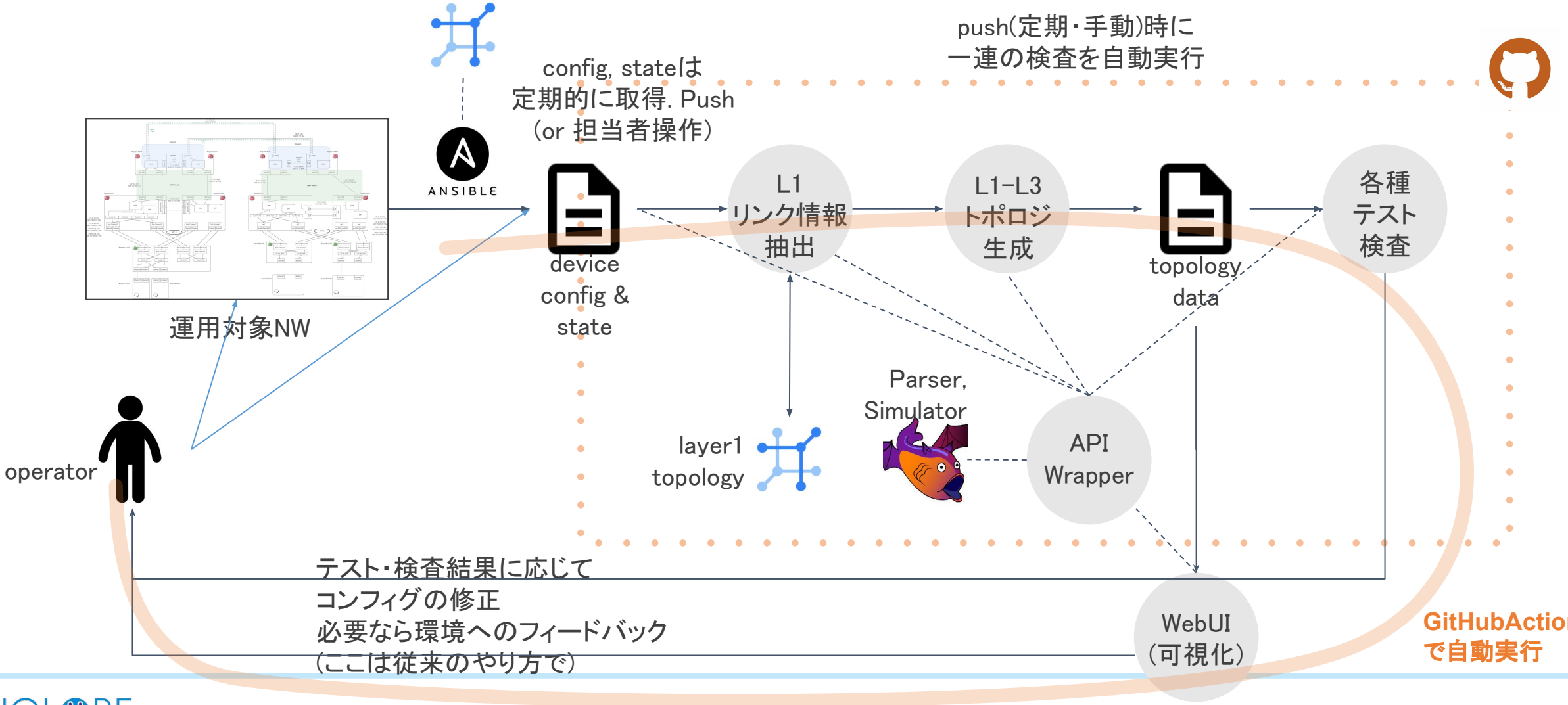
今回のシステムで実現できたこと

ノードの関係性をNWトポロジーモデルで表現することによって、
NW実機を用意せずとも
NW構成の間違いを検出できるようになった。

デモの流れ
①L1-L3トポロジとレイヤ間の関係性を把握
②ノード間の関係性をもとに静的テスト
　a: 複数IPを持つセグメント
　b: ループ検出
③経路シミュレーション

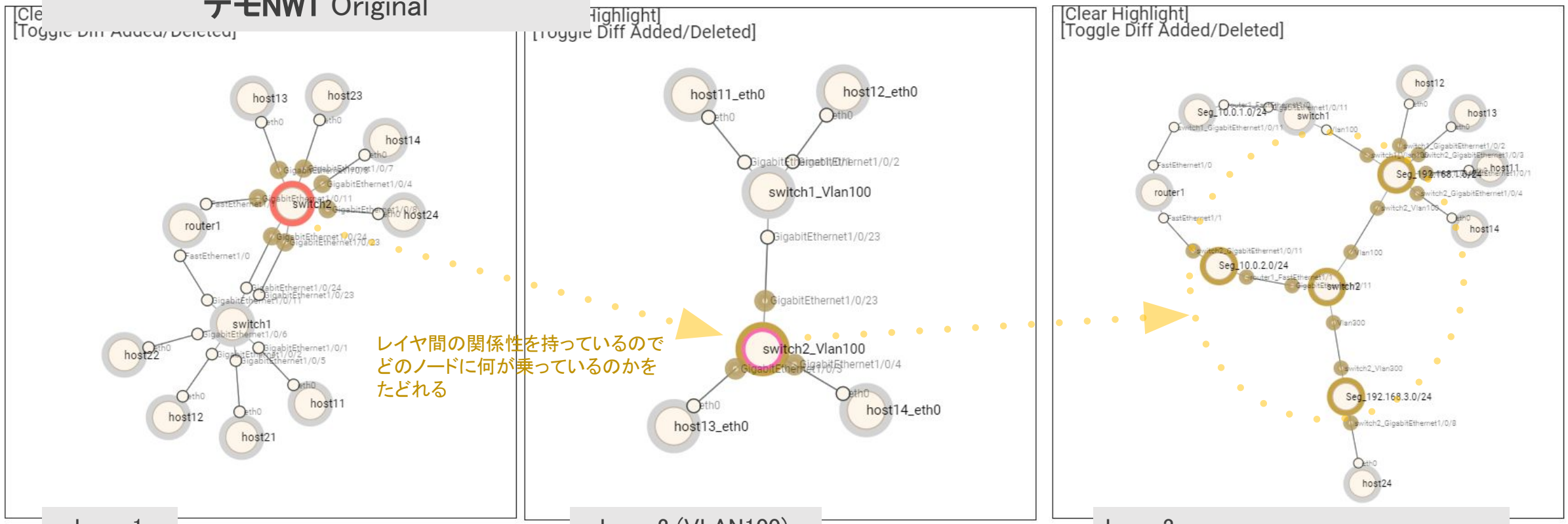
今回実装したシステム構成

Ansible NetBox Batfish



①. L1-L3トポロジとレイヤ間の関係性を把握

デモNW1 Original



レイヤ間の関係性を持っているので
どのノードに何が乗っているのかを
たどれる

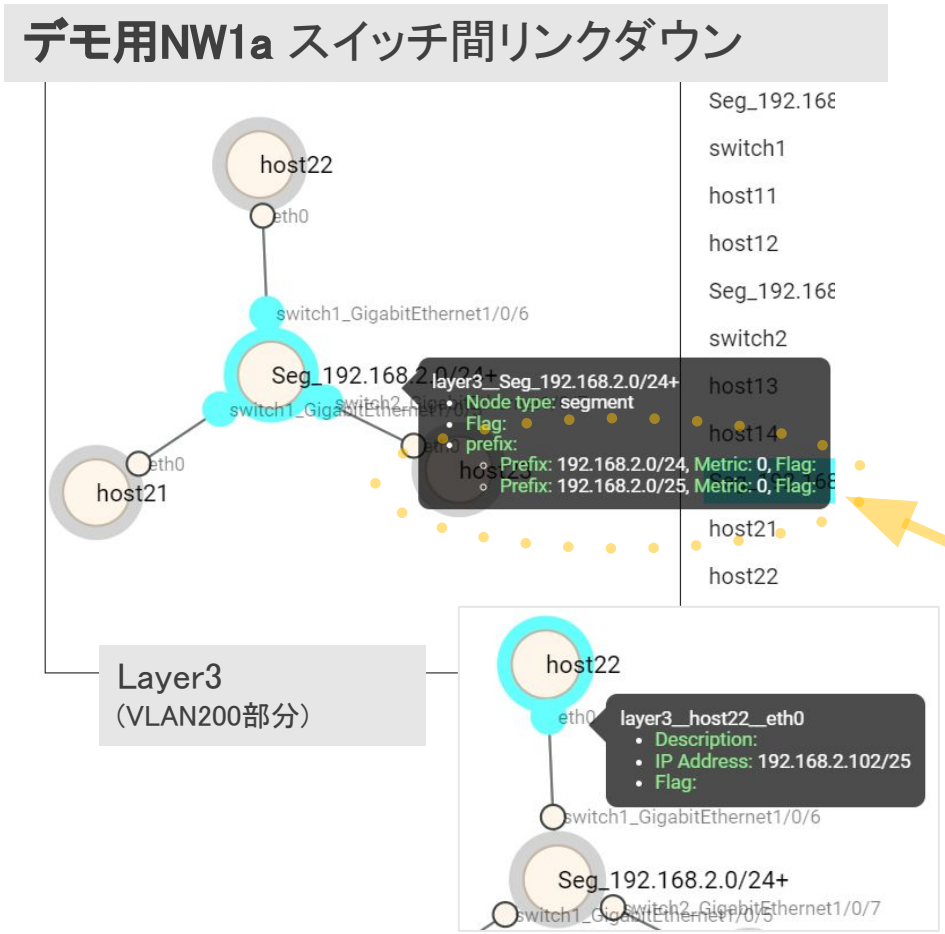
Layer1

Layer2 (VLAN100)

Layer3
(standaloneなセグメント=VLAN200以外)

L1トポロジ + Config を見て、何がどうつながっているのか
(人が設定トレースするように)たどって構成データを組み立てている
→ データ組み立て時に、接続されているポート・ノード同士の設定整合性もあわせて確認

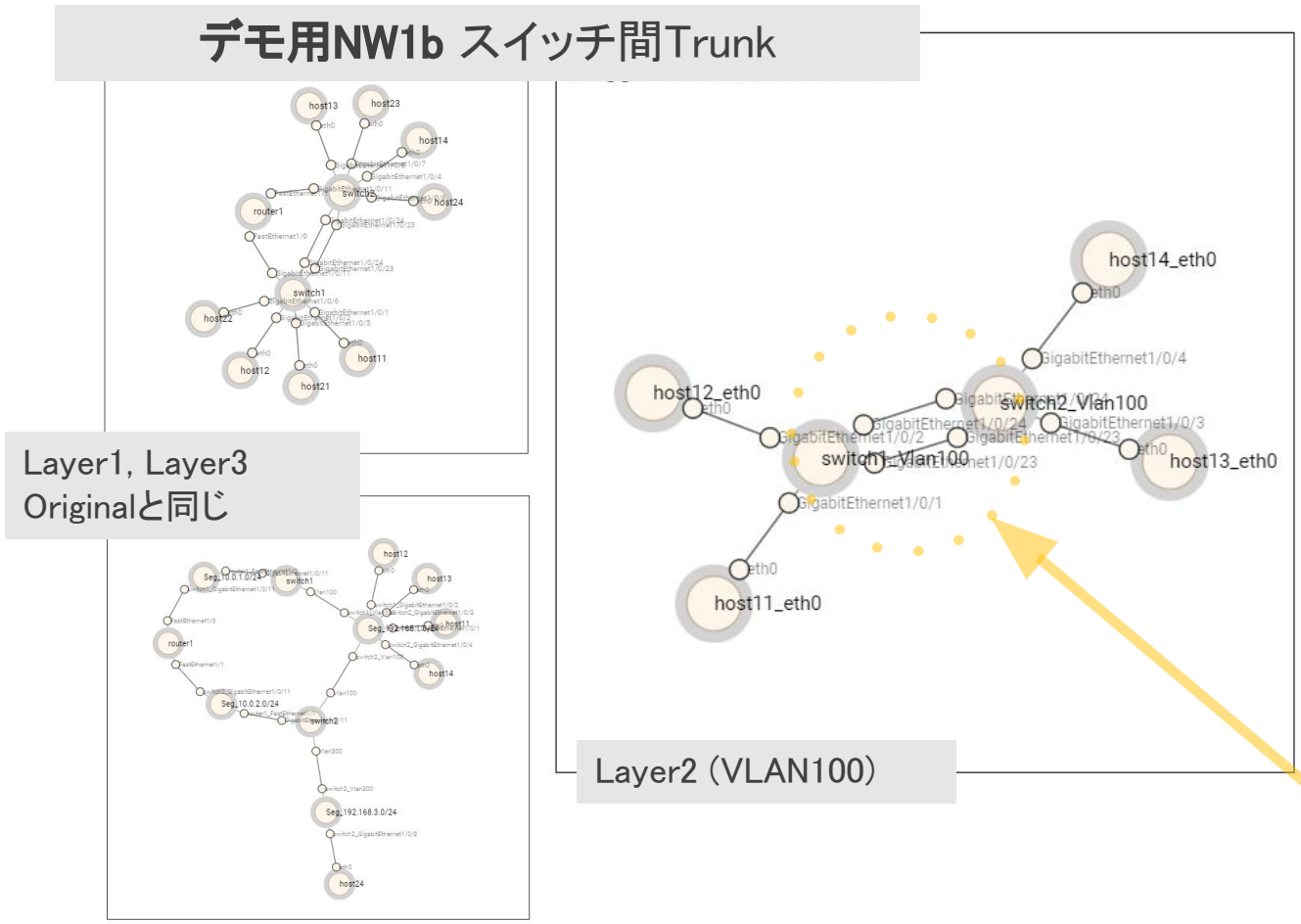
②-a. 複数IPを持つセグメント




```
hagiwara@dev02:~/ool-mddo/netomox-exp$ bundle exec ruby exe/mddo_toolbox.rb
get_subsets -f json netoviz_model/batfish-test-topology_l2l3_demo1a.json | jq
'.[] | select(.network=="layer3") | .subsets[] |
select(.flag.multiple_prefix_segments)'
{
  "elements": [
    "layer3__Seg_192.168.2.0/24+",
    "layer3__Seg_192.168.2.0/24+__switch1_GigabitEthernet1/0/5",
    "layer3__host21",
    "layer3__host21__eth0",
    "layer3__Seg_192.168.2.0/24+__switch1_GigabitEthernet1/0/6",
    "layer3__host22",
    "layer3__host22__eth0",
    "layer3__Seg_192.168.2.0/24+__switch2_GigabitEthernet1/0/7",
    "layer3__host23",
    "layer3__host23__eth0"
  ],
  "flag": {
    "multiple_prefix_segments": 1
  }
}
hagiwara@dev02:~/ool-mddo/netomox-exp$
```

セグメント名に ″+″
1つのセグメントに複数のIP prefixがある
（ここではhostのネットマスク設定ミス）

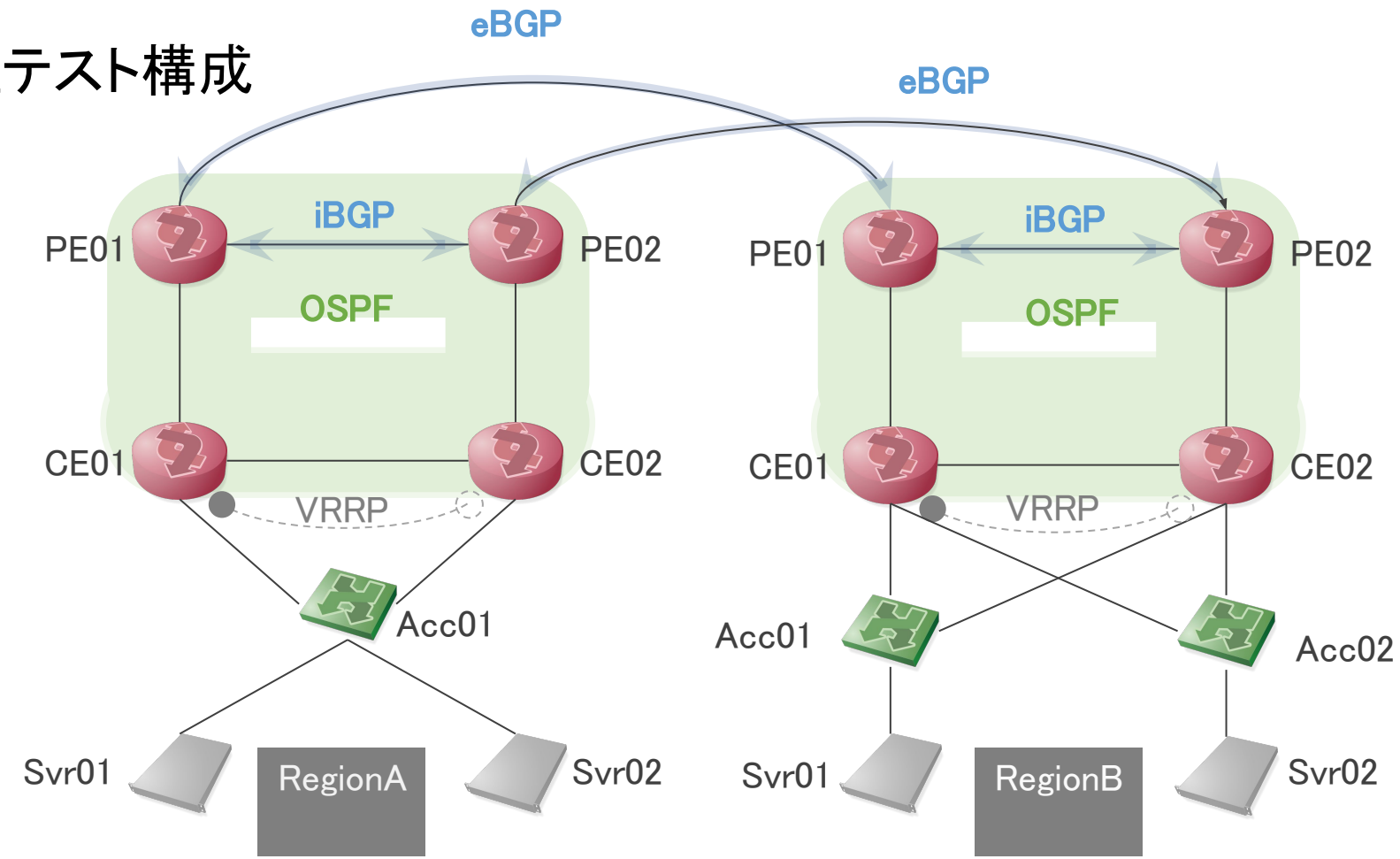
②-b. ループ検出




```
hagiwara@dev02:~/ool-mddo/netomox-exp$ bundle exec ruby
exe/mddo_toolbox.rb get_subsets -f json
netoviz_model/batfish-test-topology_l2l3_demo1b.json | jq '.[]
| select(.network=="layer2") | .subsets[] | select(.flag.loop)'
{
  "elements": [
    "layer2__switch1_Vlan100",
    "layer2__switch1_Vlan100__GigabitEthernet1/0/1",
    "layer2__host11_eth0",
    "layer2__host11_eth0__eth0",
    "layer2__switch1_Vlan100__GigabitEthernet1/0/2",
    "layer2__host12_eth0",
    "layer2__host12_eth0__eth0",
    "layer2__switch1_Vlan100__GigabitEthernet1/0/23",
    "layer2__switch2_Vlan100",
    "layer2__switch2_Vlan100__GigabitEthernet1/0/23",
    "layer2__switch2_Vlan100__GigabitEthernet1/0/3",
    "layer2__host13_eth0",
    "layer2__host13_eth0__eth0",
    "layer2__switch2_Vlan100__GigabitEthernet1/0/4",
    "layer2__host14_eth0",
    "layer2__host14_eth0__eth0",
    "layer2__switch2_Vlan100__GigabitEthernet1/0/24",
    "layer2__switch1_Vlan100__GigabitEthernet1/0/24"
  ],
  "flag": {
    "loop": true
  }
}
hagiwara@dev02:~/ool-mddo/netomox-exp$
```

③. 経路シミュレーション：L3到達性テスト

L3到達性テスト構成



※vrnetlabを利用してVM環境上で構築

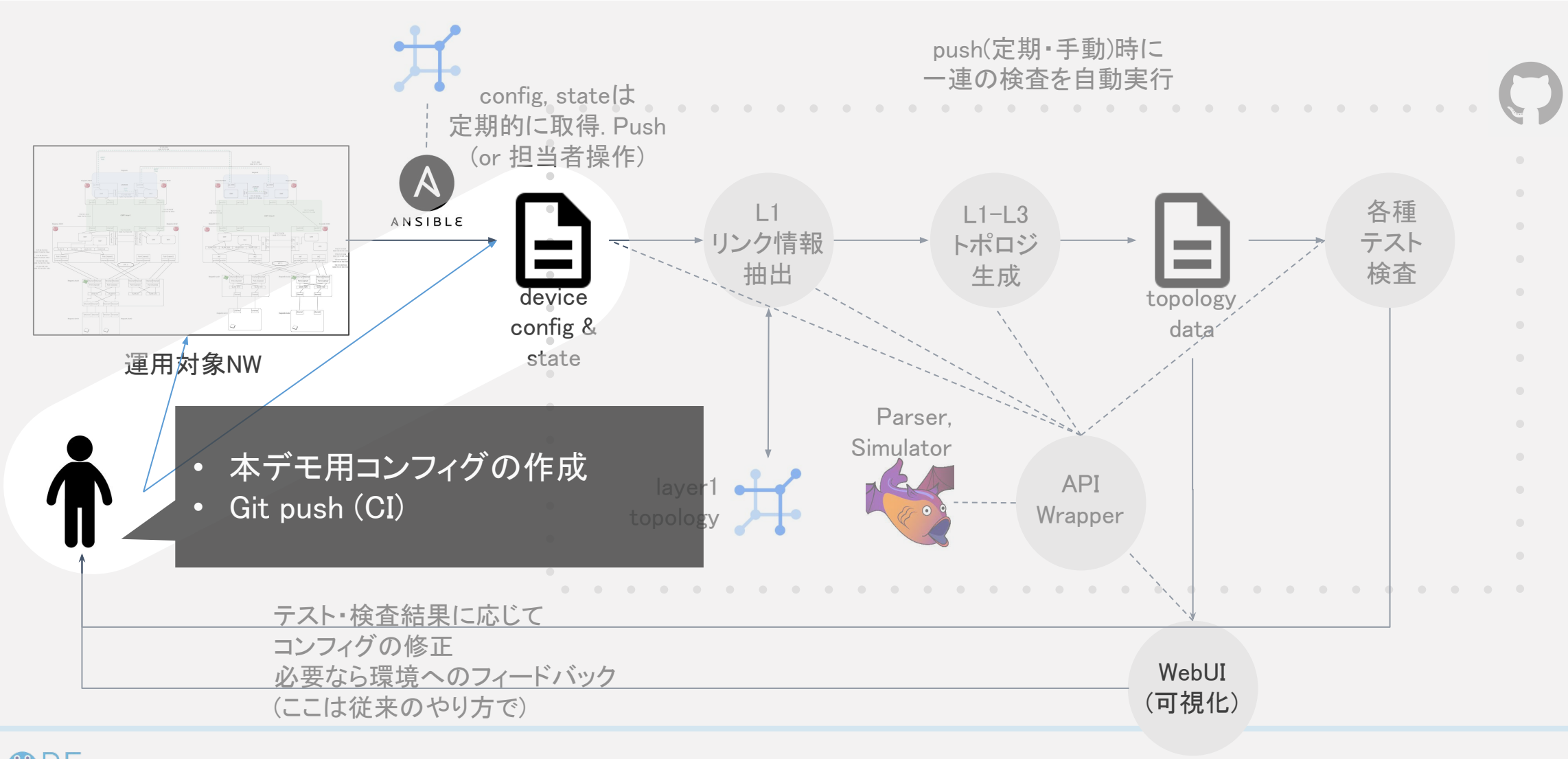
③. L3到達性テスト

回線メンテナンス時に不具合のおこる構成のコンフィグを作り、
L3到達性テストで問題が事前検出できるかを確認

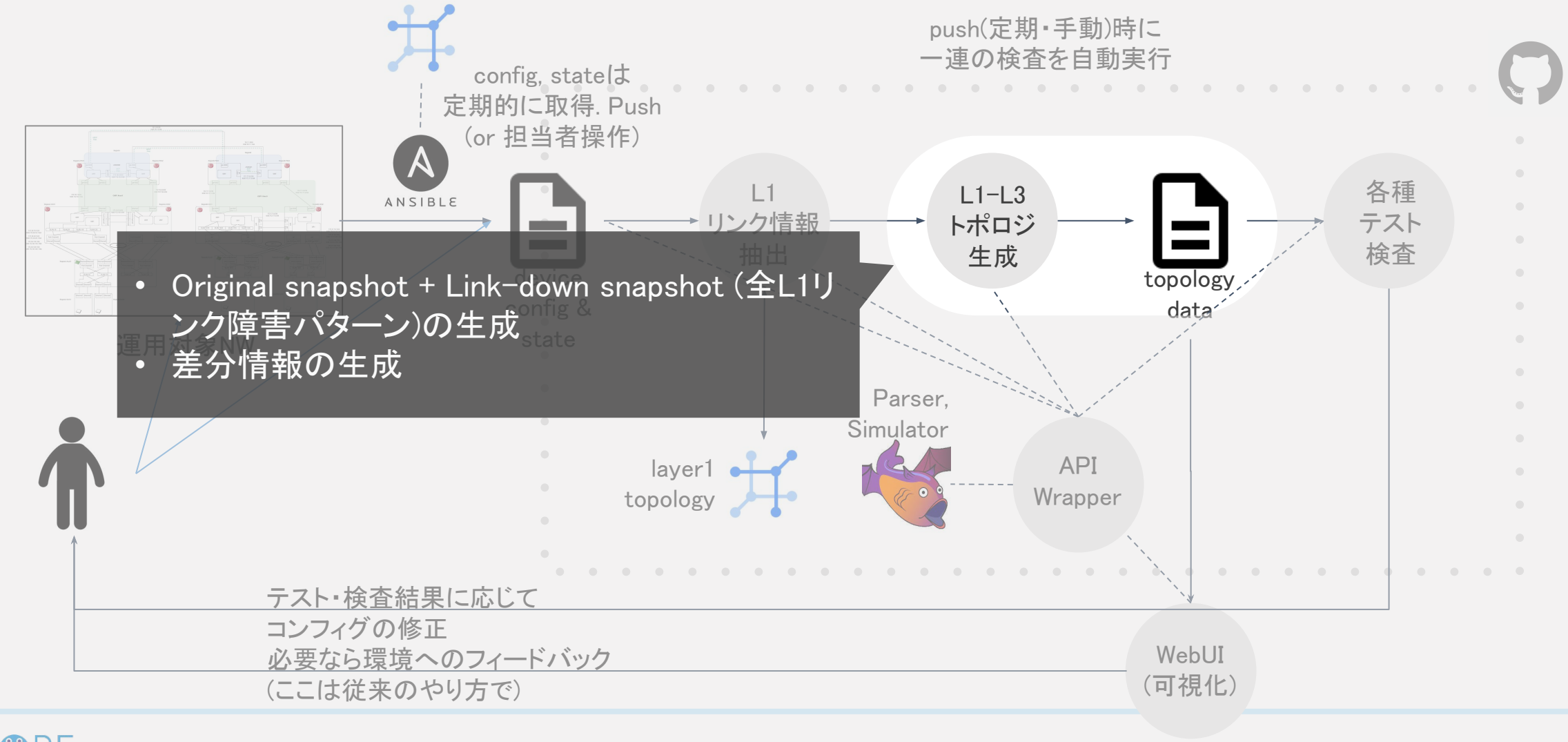
Original snapshot

Link-down snapshot

③-1: コンフィグの変更(Given)



③-2: トポロジデータの生成



③-2a: トポロジデータの生成(手動実行時)

```
hagiwara@dev02:~/ool-mddo/playground$ docker-compose up -d
hagiwara@dev02:~/ool-mddo/playground$ docker-compose run netomox-exp bash
root@b7a6d542b4e0:/netomox-exp# bundle exec rake
...
(省略)
...
bundle exec netomox diff -o /mddo/netoviz_model/pushed_configs_linkdown_mddo_network_34.json.diff
/mddo/netoviz_model/pushed_configs_drawoff_mddo_network.json /mddo/netoviz_model/pushed_configs_linkdown_mddo_network_34.json
mv /mddo/netoviz_model/pushed_configs_linkdown_mddo_network_34.json.diff
/mddo/netoviz_model/pushed_configs_linkdown_mddo_network_34.json
bundle exec netomox diff -o /mddo/netoviz_model/pushed_configs_linkdown_mddo_network_35.json.diff
/mddo/netoviz_model/pushed_configs_drawoff_mddo_network.json /mddo/netoviz_model/pushed_configs_linkdown_mddo_network_35.json
mv /mddo/netoviz_model/pushed_configs_linkdown_mddo_network_35.json.diff
/mddo/netoviz_model/pushed_configs_linkdown_mddo_network_35.json
bundle exec netomox diff -o /mddo/netoviz_model/pushed_configs_linkdown_mddo_network_36.json.diff
/mddo/netoviz_model/pushed_configs_drawoff_mddo_network.json /mddo/netoviz_model/pushed_configs_linkdown_mddo_network_36.json
mv /mddo/netoviz_model/pushed_configs_linkdown_mddo_network_36.json.diff
/mddo/netoviz_model/pushed_configs_linkdown_mddo_network_36.json
rm -f /mddo/netoviz_model/*.diff
bundle exec netomox diff -o /mddo/netoviz_model/pushed_configs_drawoff_mddo_network.json.diff
/mddo/netoviz_model/pushed_configs_mddo_network.json /mddo/netoviz_model/pushed_configs_drawoff_mddo_network.json
mv /mddo/netoviz_model/pushed_configs_drawoff_mddo_network.json.diff /mddo/netoviz_model/pushed_configs_drawoff_mddo_network.json
cp model_defs/layout/*.json /mddo/netoviz_model
root@b7a6d542b4e0:/netomox-exp#
```

- Original snapshot + Link-down snapshot（全L1リンク障害パターン）の生成
- 差分情報の生成

③-3: 構成情報を使った検査・テスト

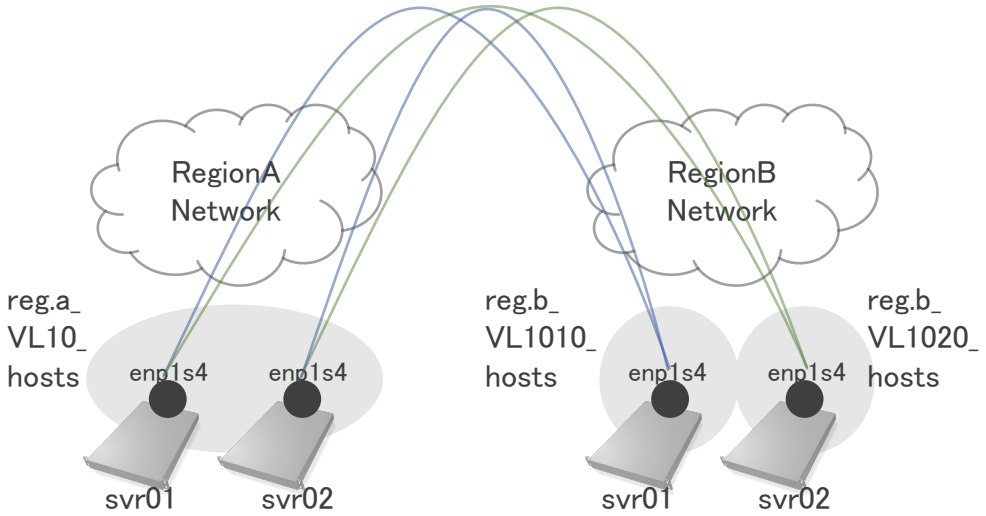
③-3a: L3到達性テスト(テストパターン定義)

```
environment:
  network: pushed_configs
  snapshot: mddo_network

groups:
  reg.a_VL10_hosts:
    - regiona-svr01__enp1s4
    - regiona-svr02__enp1s4
  reg.b_VL1010_hosts:
    - regionb-svr01__enp1s4
  reg.b_VL1020_hosts:
    - regionb-svr02__enp1s4

patterns:
  - [reg.a_VL10_hosts, reg.b_VL1010_hosts]
  - [reg.a_VL10_hosts, reg.b_VL1020_hosts]
```

グループ定義
同一ポリシで扱うEndpoint(IPを持つインタフェース)の集合

テストパターン定義
どのグループ間で
L3到達性テストをするか?

デモでは
Source group → Destination group の組み合わせを生成
グループ間の双方向全組み合わせなど、
テスト粒度に応じて実装は考えられる



③-3b: L3到達性テスト(Linkdown)

Link-down snapshots

```
root@782dac92ba7c:/netomox-exp# bundle exec ruby exe/mddo_toolbox.rb test_reachability exe/traceroute_patterns.yaml -n pushed_configs_linkdown -r
(省略)

      test: pushed_configs_linkdown/mddo_network_19 <No.19: down RegionA-PE01[ge-0/0/2] <=> RegionA-CE01[Ethernet1] in layer1>:                F
================================================================================================================================================
=
Failure: test: pushed_configs_linkdown/mddo_network_19 <No.19: down RegionA-PE01[ge-0/0/2] <=> RegionA-CE01[Ethernet1] in
layer1>(TestTracerouteResult::PATTERN: reg.a_VL10_hosts->reg.b_VL1020_hosts::CASE: regiona-svr02[enp1s4](172.30.10.101) ->
region-svr02[enp1s4](172.31.20.100)): assert_block failed.
/home/hagiwara/ool-mddo/netomox-exp/exe/reach_test/test_traceroute_result.rb:14:in `block (6 levels) in <class:TestTracerouteResult>'
     11:         sub_test_case "CASE: #{test_case['case'][0]} -> #{test_case['case'][1]}" do
     12:           test_case['traceroute'].each do |trace|
     13:             test "#{trace[0]}/#{trace[1]} <#{trace[2]}>" do
  => 14:               assert_block { %w[ACCEPTED DISABLED NEIGHBOR_UNREACHABLE].include?(trace[3]) }
     15:             end
     16:           end
     17:         end
================================================================================================================================================
=
: (0.000920)
      test: pushed_configs_linkdown/mddo_network_20 <No.20: down RegionB-PE01[ge-0/0/1] <=> RegionB-PE02[ge-0/0/1] in layer1>:
.: (0.000056)

(省略)

Finished in 0.02493022 seconds.
------------------------------------------------------------------------------------------------------------------------------------------------
-
144 tests, 144 assertions, 4 failures, 0 errors, 0 pendings, 0 omissions, 0 notifications
97.2222% passed
------------------------------------------------------------------------------------------------------------------------------------------------
-
```

Failしたテストケースがある
(L3通信うまくいかなかった)

4/144 で Fail
144 = 36 snapshots(links) * 4 flows

③-3c: L3到達性テスト(Linkdown)

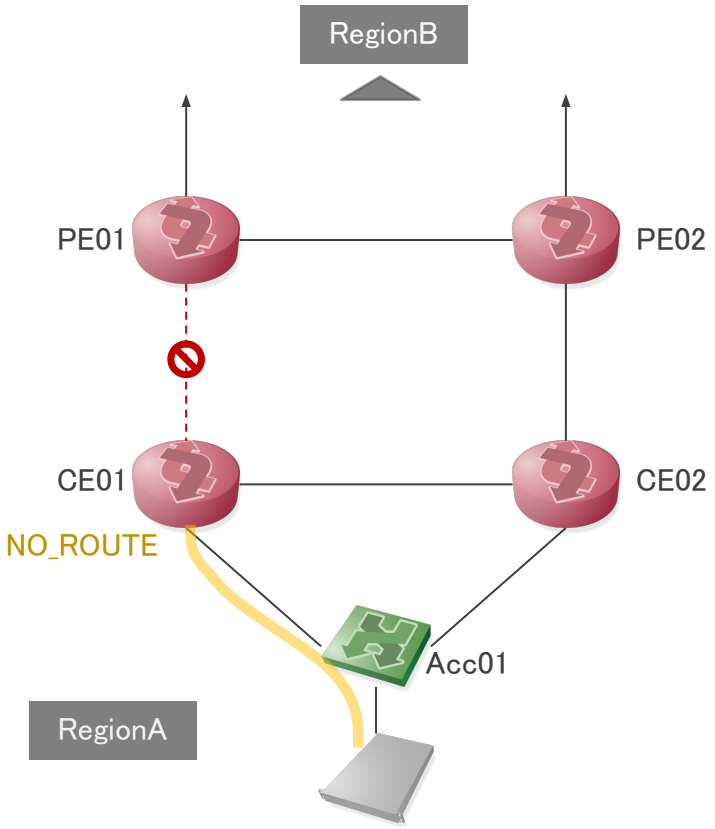
テスト結果(サマリ/CSV)で失敗したテストケースを検索

```
root@782dac92ba7c:/netomox-exp# csvtool col 6-8 pushed_configs_linkdown.test_summary.csv  | column -s, -t | egrep -vi '(accepted|disabled|neighbor_unreach)'
Description                                                                 Deposition           Hops
No.19: down RegionA-PE01[ge-0/0/2] <=> RegionA-CE01[Ethernet1] in layer1    NO_ROUTE             regiona-svr01[enp1s4]->regiona-ce01[Vlan10]
No.19: down RegionA-PE01[ge-0/0/2] <=> RegionA-CE01[Ethernet1] in layer1    NO_ROUTE             regiona-svr02[enp1s4]->regiona-ce01[Vlan10]
No.19: down RegionA-PE01[ge-0/0/2] <=> RegionA-CE01[Ethernet1] in layer1    NO_ROUTE             regiona-svr01[enp1s4]->regiona-ce01[Vlan10]
No.19: down RegionA-PE01[ge-0/0/2] <=> RegionA-CE01[Ethernet1] in layer1    NO_ROUTE             regiona-svr02[enp1s4]->regiona-ce01[Vlan10]
root@782dac92ba7c:/netomox-exp#
```

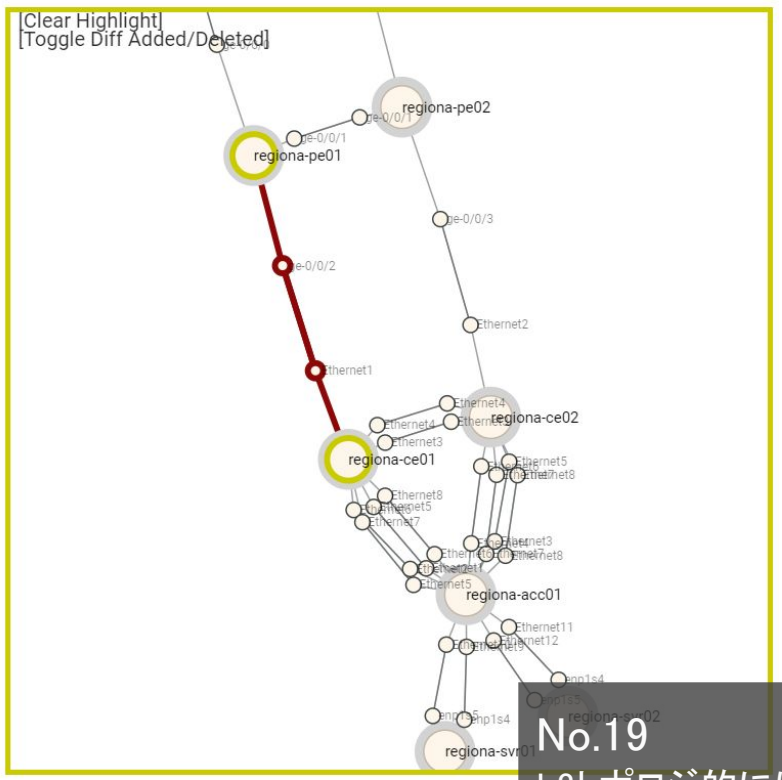
テストケース No.19 でFail
RegionA-PE01～CE01間のリンクダウン

Hops列
Server → CE (default gateway) で止まっている?

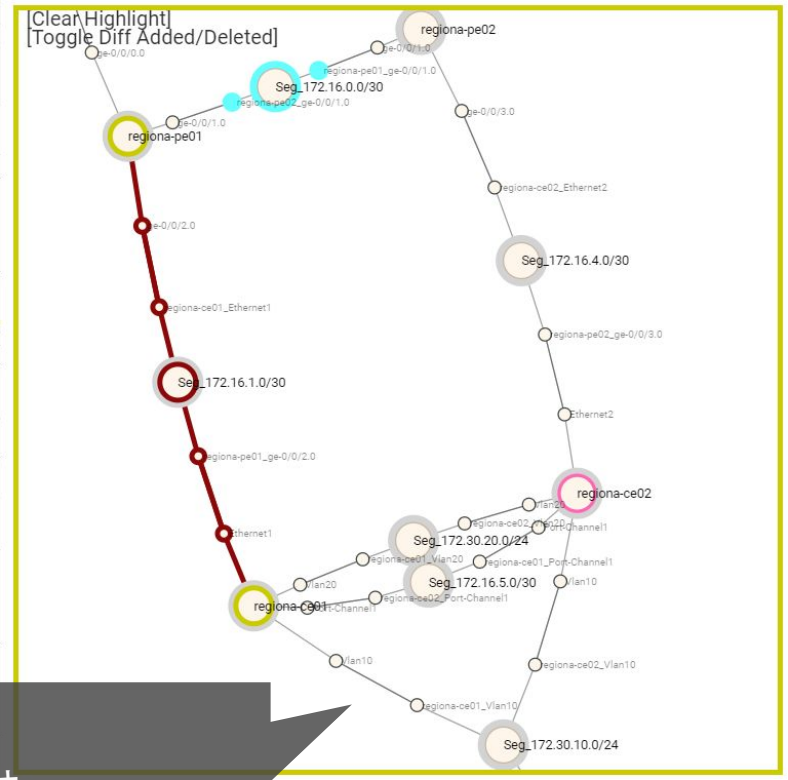
③-4a: 各レイヤの構造チェック(No.19)



Layer1



Layer3



No.19
L3トポロジ的には
迂回経路がありそう
なぜCE02側に迂回しない?

③-4b: デバイスの状態確認(No.19)

- 検証環境で実際に調査する
- Batfish上で調査する

＋Config確認

```
hagiwara@dev02:~/ool-mddo/playground$ docker-compose exec batfish-wrapper python -i
(省略)
>>> from pybatfish.client.session import Session
>>> bf = Session()
>>> bf.set_network('pushed_configs_linkdown')
'pushed_configs_linkdown'
>>> bf.set_snapshot('mddo_network_19')
'mddo_network_19'
>>> bf.q.routes(nodes='regiona-ce01').answer().frame()
          Node      VRF          Network     Next_Hop_IP Next_Hop_Interface    Protocol Metric Admin_Distance   Tag
0  regiona-ce01  default    172.16.5.0/30  AUTO/NONE(-1l)      Port-Channel1   connected      0              0  None
1  regiona-ce01  default   172.30.10.0/24  AUTO/NONE(-1l)             Vlan10   connected      0              0  None
2  regiona-ce01  default   172.30.20.0/24  AUTO/NONE(-1l)             Vlan20   connected      0              0  None
3  regiona-ce01     vrf1  172.30.110.0/24  AUTO/NONE(-1l)            Vlan110   connected      0              0  None
4  regiona-ce01     vrf1  172.30.120.0/24  AUTO/NONE(-1l)            Vlan120   connected      0              0  None
>>> bf.q.ospfEdges(nodes='regiona-ce01').answer().frame()
Empty DataFrame
Columns: [Interface, Remote_Interface]
Index: []
>>> bf.q.vrrpProperties(nodes='/regiona-ce*/', interfaces='/Vlan10/').answer().frame()
             Interface Group_Id Virtual_Addresses  Source_Address Priority Preempt
0  regiona-ce02[Vlan10]        1   ['172.30.10.3']  172.30.10.2/24      100    True
1  regiona-ce01[Vlan10]        1   ['172.30.10.3']  172.30.10.1/24      110    True
>>>
```

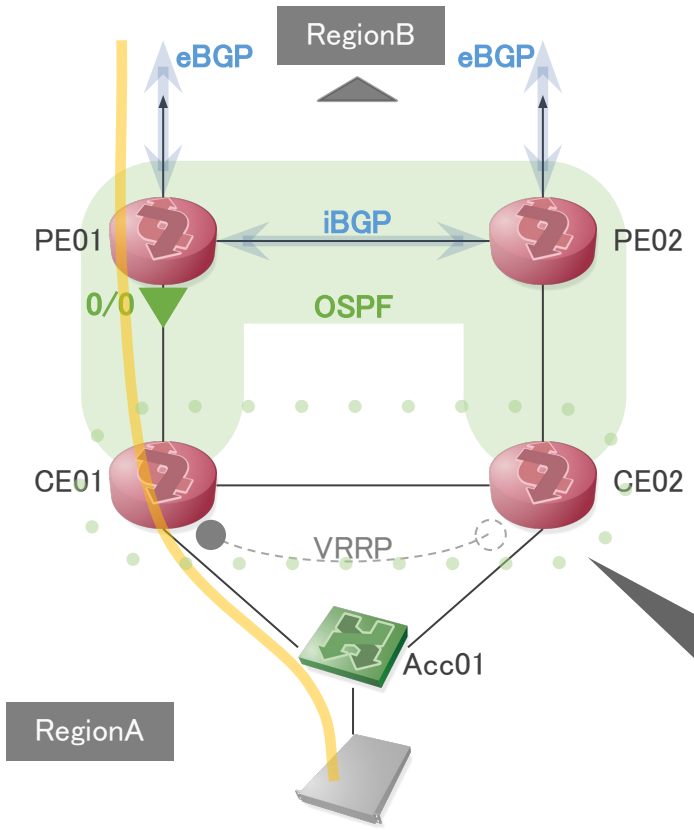
No.19

CE01
CE02からOSPFで経路もらっていると思ったらない…

OSPF Neighborがない

VRRP Master = CE01

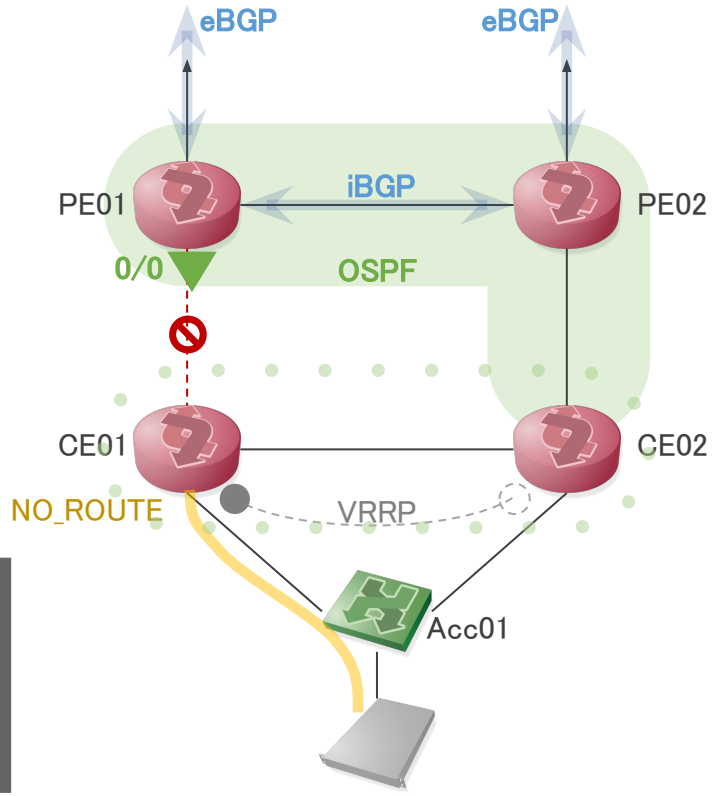
③-4c: 原因特定



Original snapshot

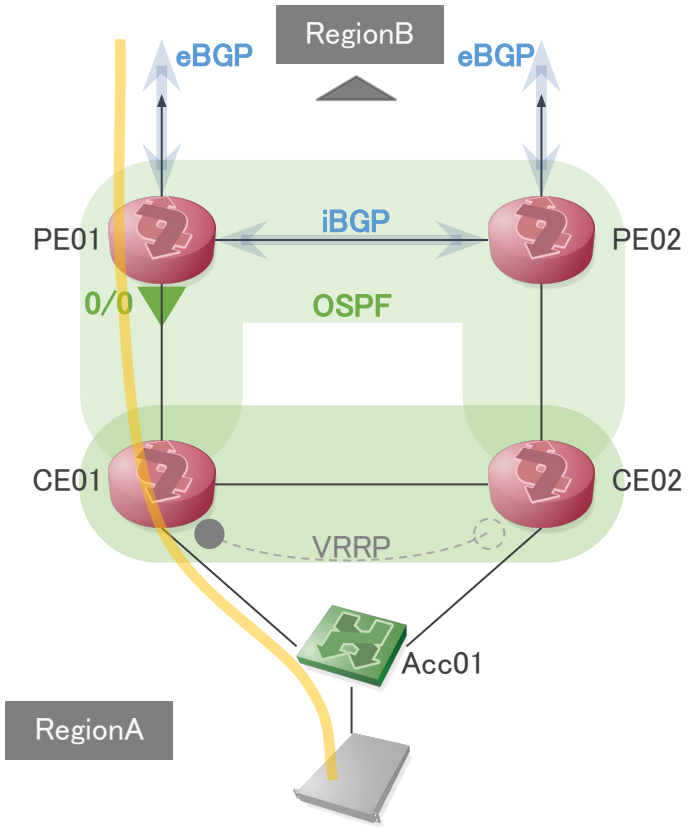
設定モレ
CE間のOSPF設定を入れ忘れていた

本来はCE01-02間もOSPF Area
CEがuplinkしかAreaに入っていないので
切れるとRegion内経路がはいってこない



Link-down snapshot (No.19)

③-5: 修正



Original snapshot
（修正）

CE間にOSPF設定を追加

```
diff --git a/mddo_network/configs/RegionA-CE01_config.txt
b/mddo_network/configs/RegionA-CE01_config.txt
index ff715ba..4ee5958 100644
--- a/mddo_network/configs/RegionA-CE01_config.txt
+++ b/mddo_network/configs/RegionA-CE01_config.txt
@@ -29,6 +29,7 @@ interface Port-Channel1
     ip address 172.16.5.1/30
     ipv6 address fc00:172:16:5::1/64
     port-channel min-links 1
+    ipv6 ospf 1 area 0.0.0.0
  !
  interface Port-Channel2
     load-interval 30
@@ -165,6 +166,7 @@ router ospf 1
     passive-interface default
     no passive-interface Ethernet1
     no passive-interface Ethernet2
+    no passive-interface Port-Channel1
     network 0.0.0.0/0 area 0.0.0.0
     max-lsa 12000
  !
```

再度テストを実施

③-6: 修正後の再チェック

修正したコンフィグでトポロジデータ再作成

```
root@b7a6d542b4e0:/netomox-exp# bundle exec rake

(省略)

root@b7a6d542b4e0:/netomox-exp# bundle exec ruby exe/mddo_toolbox.rb test_reachability exe/traceroute_patterns.yaml -n pushed_configs_linkdown -r

(省略)

Finished in 0.016242604 seconds.
----------------------------------------------------------------------------------------------------------------------------------------
-
144 tests, 144 assertions, 0 failures, 0 errors, 0 pendings, 0 omissions, 0 notifications
100% passed
----------------------------------------------------------------------------------------------------------------------------------------
-
8865.57 tests/s, 8865.57 assertions/s
hagiwara@dev02:~/ool-mddo/netomox-exp$
```

テスト再実行

テストが全部通った

今回実現できた成果・効果

- 人手ではできないテストの実現
  - 網羅テスト：範囲(NW全体に対して)・組み合わせ(パターン)
    - 「やってみて初めて気づいた」の回避
  - 自動化、CI/テストドリブンな手法のとりこみ

- こうした検討トライアルを分単位で実施可能
  - 環境サイズ(ノード数)・テストパターン数に依存する
    - 今回のデモの規模・内容だと数分～
  - 検討段階でも試せる・実機/実環境がなくても試せる
    - 「やってみないとわからない」の回避
  - 実機の調整→設定→動作確認よりもイテレーション間隔を短縮できる

課題点

- Batfish依存問題
  - 市販製品導入時の制約事項と同様の制約が起きうる
  - Config parserとしての機能は比較的簡単に代替できる
  - Network simulationをどうするか (IPv6対応問題)
    - Write方向技術との組み合わせ…実働NWからの状態取り込み
  - スケーラビリティ
- できあがったトポロジデータ、Query結果検証の難しさ
- データの読み取り・分析→デプロイまでのサイクル
  - 動的な検証環境との併用・組み合わせ、stateの取り込み
- 実ネットワークの構造や機能の取り込み・モデル化
  - 動的ルーティングプロトコル、L4以上の機能/ノード、仮想ネットワーク

今後の発展性

- ネットワークリソースの追加・変更・削除
  - トポロジの変更・変動, VLAN等仮想リソース, Overlay
  - Peer/Neighbor, 経路制御設定, Policy/Filter
- 障害試験
  - CI, カバレッジの拡大・向上
  - 模擬・シミュレーション, 経路切り替えなどの手順確認等
- 特定のトラフィック(フロー)がどこを経由しているかの確認
  - 実態・リアルタイムステートの把握(STPやVRRPなど)
  - TEにおけるベストパス・セカンダリパスの確認
    - とりうるパスを優先順位別に確認・選択

最後に

沖縄オープンラボラトリのModel Driven Network DevOps Projectは一緒に活動してくれるプロジェクトメンバーを募集しています。

↓募集フォームは以下になります。↓
https://www.okinawaopenlabs.org/mdnd

また、
直接話を聞きたい方は気軽にPJメンバーへ話しかけてください。
よろしくお願いいたします。

付録

・MDDOのGithubリポジトリ
https://github.com/ool-mddo

・テックブログ
今回の内容を詳しく解説した3社でのブログリレー

- BIGLOBE
  https://style.biglobe.co.jp/entry/2022/03/30/090000?fbclid=IwAR35ZDUJUyURx1aOnY5lmZmG6f0FhkXLU-Hcitp7nCue3r_7-dD-OgS--VM
- NTTコミュニケーションズ
  https://engineers.ntt.com/entry/2022/03/31/090000?fbclid=IwAR3oX2WPgUVJMEJEPgklMQUGdsNMfBbbIrVtvSdLOikVno75RV8agiw52YY
- TIS
  https://www.tis.jp/special/platform_knowledge/nw02/

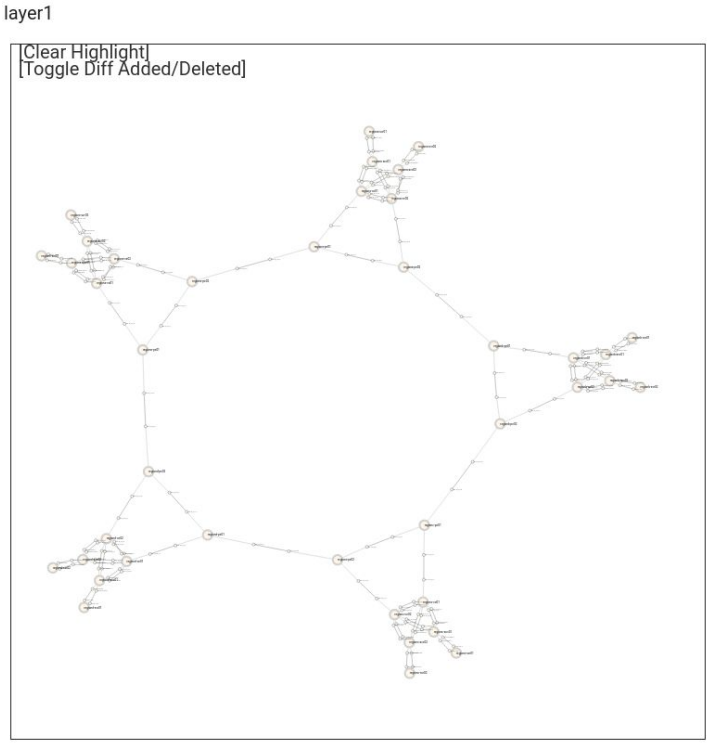
ご清聴ありがとうございました

Appendix

多リージョン試験（商用規模での動作確認）: NWの規模と処理時間

NWサイズによるパフォーマンス変化

Region	NW Device	Host	Link
2	11	4	36
5	30	10	90
10	60	20	180
20	120	40	360
40	240	80	720

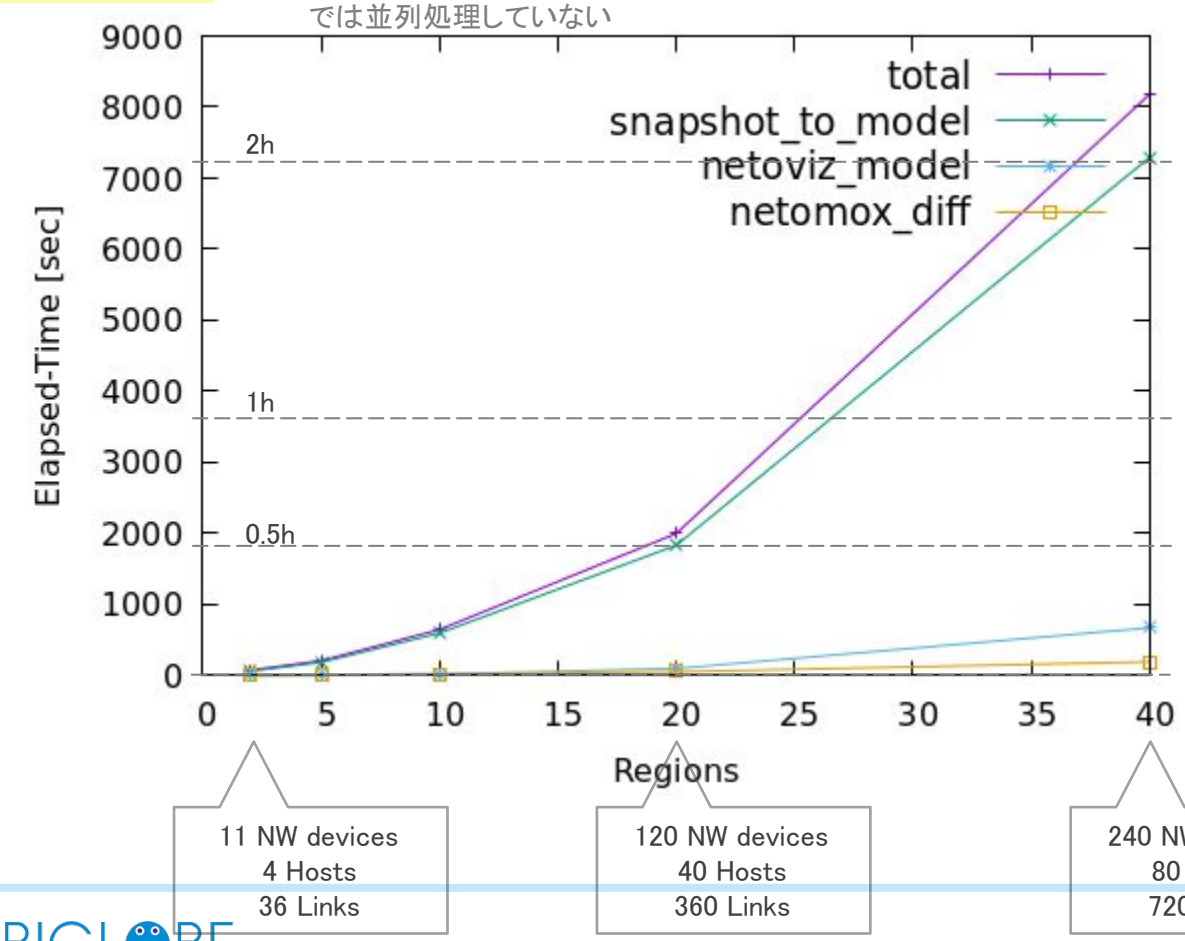


5リージョン構成(L1)

トポロジデータ作成にかかる時間
(リンク障害パターンのトポロジ…L1リンク数分を含む)

データ生成
(静的検査)
にかかる時間

Intel Xeon 2.10GHz 16C/32T
ただしパーサ/シミュレータ(batfish)メモリ使用量を抑えるため
データ生成セクション(snapshot_to_model)
では並列処理していない

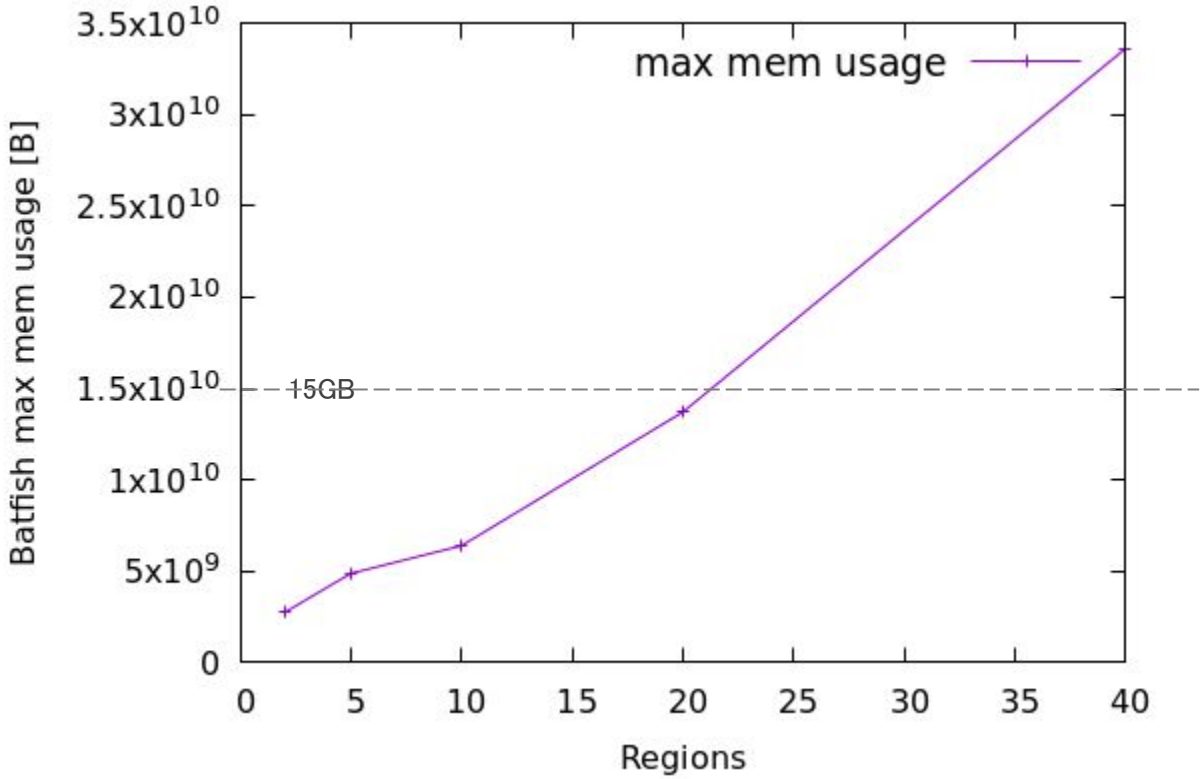


11 NW devices
4 Hosts
36 Links

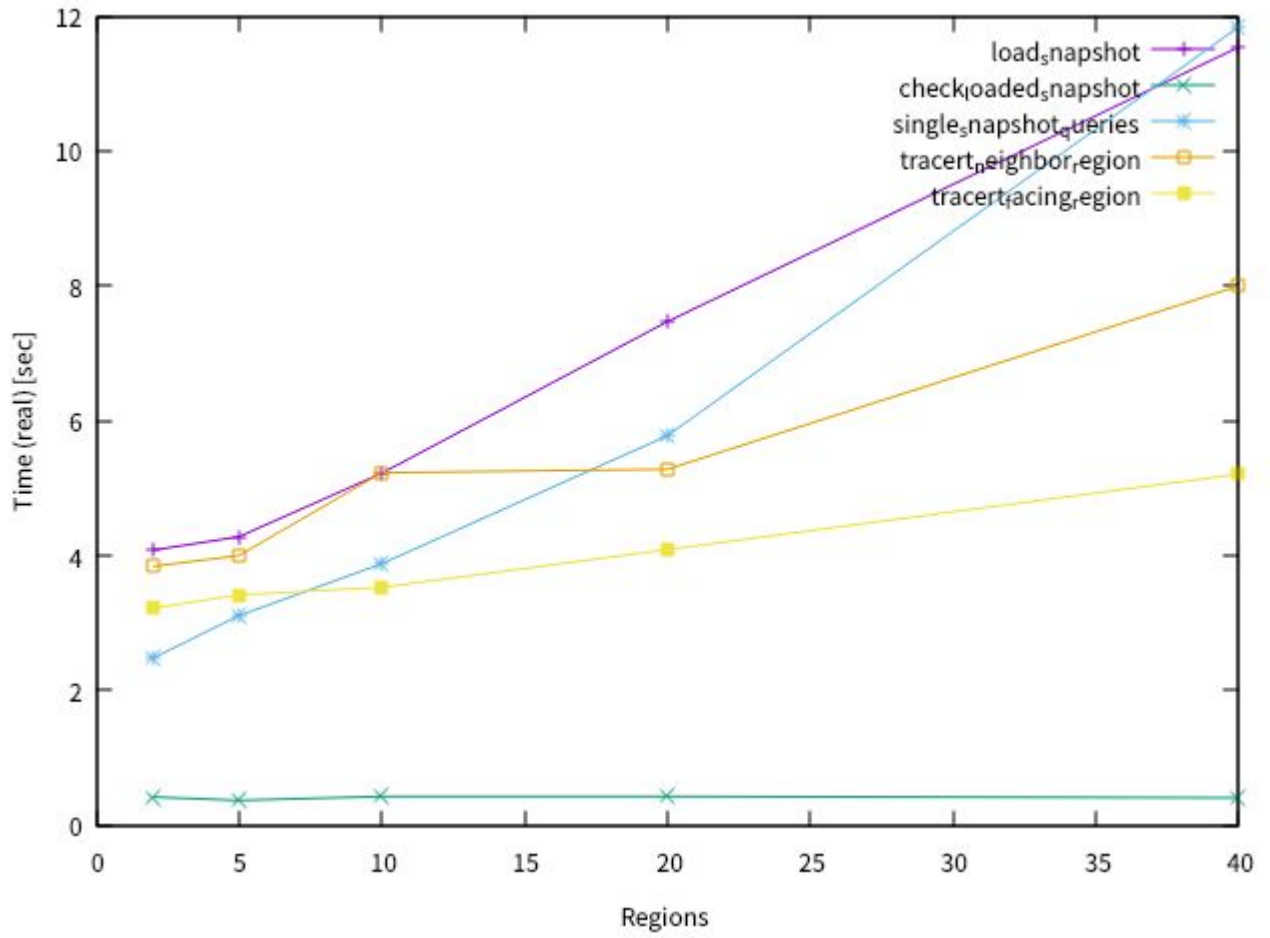
120 NW devices
40 Hosts
360 Links

240 NW devices
80 Hosts
720 Links

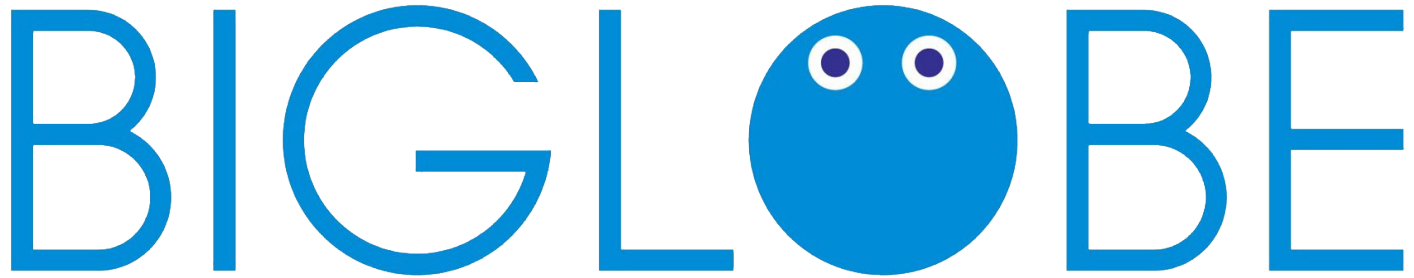
パーサ/シミュレータ(batfish)
最大メモリ使用量



処理単体にかかる時間



- load_snapshot:
  スナップショットのロード
- check_loaded_snapshot:
  ロードされているスナップショットのリスト取得
- single_snapshot_queries:
  スナップショット(単体)に対して各種queryを実行してCSVに落とす
- tracert_neighbor_region:
  RegionA–B (隣接間のL3通信シミュレーション(traceroute)
- tracert_facing_region:
  RegionA–最も遠いAS間のL3通信シミュレーション(traceroute)


BIGLOBE