Contrail Enterprise Multicloud (CEM) General Presentation

2020年7月

ジュニパーネットワークス株式会社

Juniper Networks | Engineering Simplicity

AGENDA

- MultiCloudへの取り組み
- Contrail Enterprise Multicloud 概要
  - Contrail Networking
  - Contrail Security
  - Contrail Multi Cloud
  - Contrail Fabric
  - Contrail Insights(AppFormix)
- CEMアーキテクチャ
- CEM+OpenStack
- CEM+K8S
- CEM+vCenter
- Contrail Enterprise Multicloud 機能詳細
  - Contrail Networking
  - Contrail Fabric (Contrail Fabric Managerジェネプレ参照)
- CEM構成例
- CEM Release Policy

CONTRAIL 製品カテゴリ

本資料対象

Contrail Enterprise MultiCloud (CEM)

Contrail Command
- UI for CEM

Contrail Networking
- Controller
- vRouter

Contrail Security
- インテントベース Firewall
- 分散型 Firewall (L4)
- cSRX for Contrail vRouter(L7)

Contrail Fabric
- Underlay Network Automation/Management

Contrail MultiCloud
- AWS/GCP/Azure接続
- MultiCloud Gateway

Contrail Insights (旧Appformix)
- Monitoring / Analytics
- Underlay/Overlay Correlation

Contrail Service Orchestrator (CSO)
- SD-WAN/SD-LAN クラウド型 Controller or SD-WAN/SD-LAN オンプレ型 Controller
- SRX/NFX CPE
- MIST WiFi

Contrail Cloud
- Redhat OpenStack/Ceph/Satelite と Contrail Networking/Security/Insight の 検証済みパッケージソリューション

Contrail HealthBot
- ネットワーク自動運用ソリューション
- ネットワーク機器のリアルタイム監視・診断・可視化
- ワークフローのプログラム化・カスタマイズ

Contrail Insights (旧Appformix)
- Monitoring / Analytics
- Underlay/Overlay Correlation

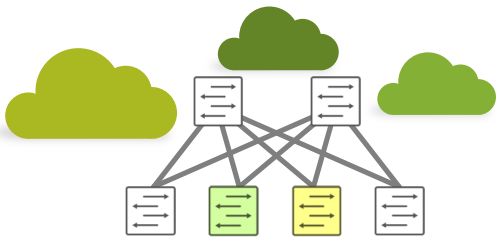
MultiCloudへの取り組み

マルチクラウド実現にむけた
ヘテロジーニアス環境へのオープン・システム・アプローチ

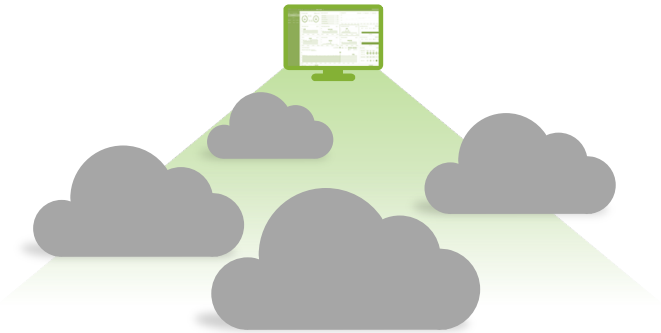
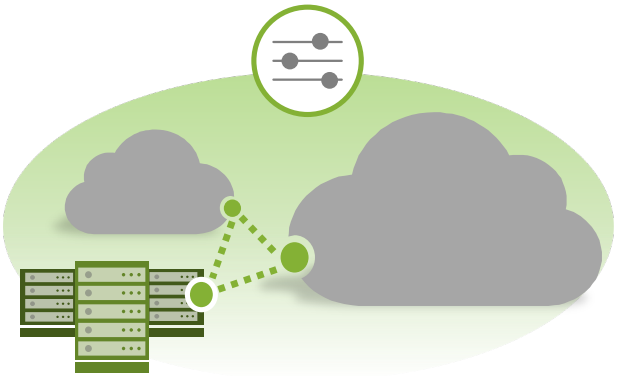
VUCA時代には**ビジネス・アジリティ**が最大の武器

オープン・エコシステムを採用することで常に最適なサービス提供が可能に

tungstenfabric

ANY CLOUD	ANY WORKLOAD	ANY DEPLOYMENT
OPENSHIFT, kubernetes, vmware, NUTANIX, openstack, aws, Google Cloud Platform, Azure		
Private & Public Cloud	BMS, VM, Container Physical and Vitual Network	Greenfield or Brownfield, Single- or Multi-vendor

マルチクラウド実現にむけた
可視化、セキュリティ、自動化

可視化	セキュリティ	自動化
		
Compute, Virtual Instance Network device, Flow Analytics & Telemetry	統合セキュリティオーケストレーション L4-L7 NGFW	マルチクラウド・コネクティビティ オーバーレイ・仮想ネットワーク アンダーレイ・プロビジョニング

自動化： 仮想NW、物理NW、DCI、マルチクラウド環境におけるコネクティビティの自動化

セキュリティ： ATPよびL7セキュリティのエンフォースメント、統合セキュリティポリシーオーケストレーション

可視化： UX, Hardware, Software, Instanceのリアルタイム可視化および分析

MULTICLOUD実現への 5-STEP フレームワーク

現在の取り組み

人的な変革 → プロセスの変革 → テクノロジーの変革

1 Device Led

Traditional

- 従来型のマニュアル運用
- 境界セキュリティ

2 Architecture Led

Simplified

- IP ファブリック
- テレメトリ分析
- セキュリティ脅威検知
- クラウド基盤の自動化

3 Operations Led

Software-Defined

- EVPN-VXLAN導入
- クラウドオーケストレータとのSDNの連携
- 分析と検証の自動化
- IaaC

4 Business Led

Multi-Domain

- DCIも含めたマルチクラウド環境におけるEVPN-XVLAN導入
- クラウドをまたいだマイクロセグメンテーション
- 脅威に対する自動プロテクション
- CICDとNetOps
- イベント発生時の自動ヒーリング

5 Customer Led

Secure and Automated Multicloud

- 統合化されたオーケストレーション
- 統合化された分析、モニタリングツール
- AI for IT
- 予兆検知と自動対処
- サービスレベルの可視化によるNREの実現

※NRE: Network Reliability Engineering

MULTICLOUD ポートフォリオ

可視化

＋

セキュリティ

＋

自動化

＋

オープン

Software

Security Director/
Policy Enforcer

ATP

Contrail
Enterprise
Multicloud

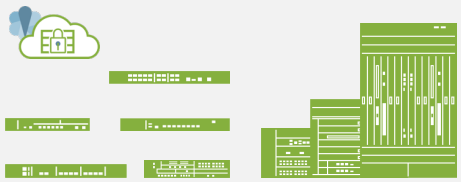
Contrail
HealthBot

AppFormix

Juke

Junos OS

Secure Edge
SRX Series NG-FW



vSRX, cSRX, SRX300-5000s

Switching
QFX Series

QFX5000s-10000s

Routing
MX Series

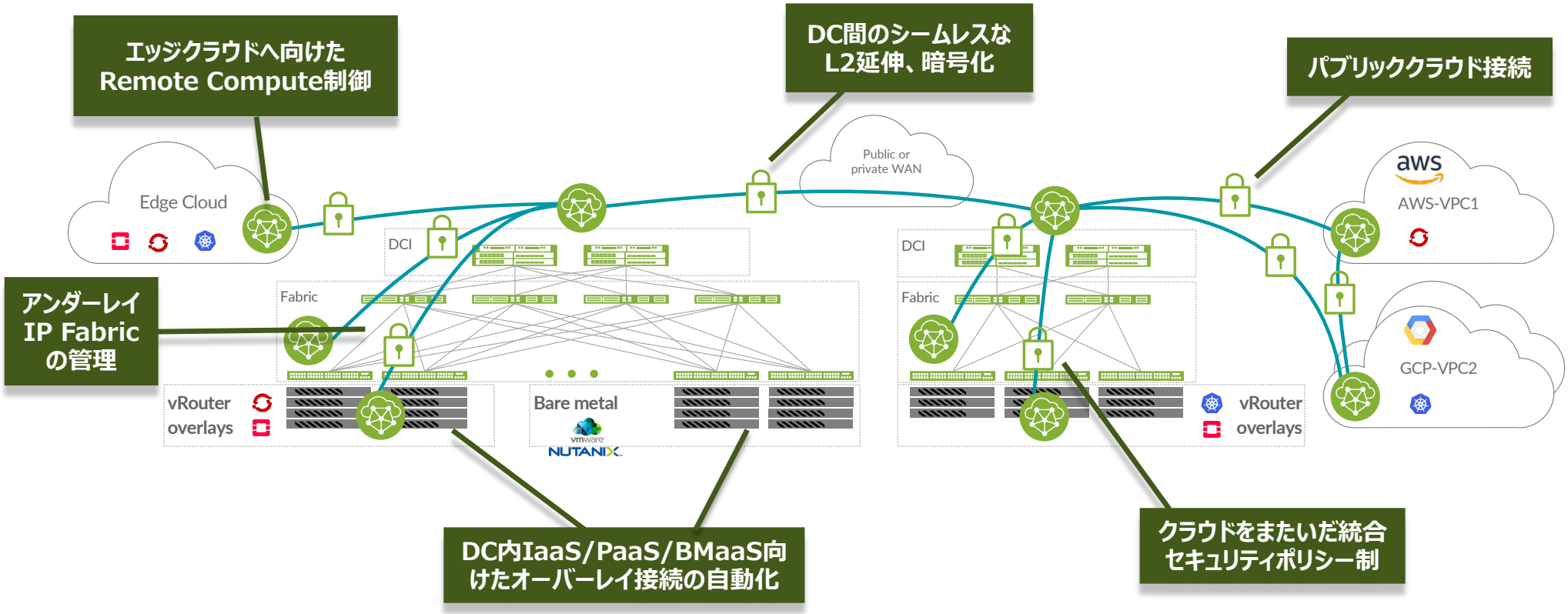
vMX, MX204, 10003 and 240+

Alliances Partners

redhat.
CANONICAL
OPENSHIFT
openstack.
kubernetes
NUTANIX
aws
Azure
STACKPATH
Google Cloud Platform
IBM Cloud

マルチクラウドにおけるEND-TO-END制御

クラウドのワークロードにおけるロケーション・ダイバーシティが進むことでネットワークやセキュリティの重要度が増してきている



CONTRAIL ENTERPRISE MULTICLOUD(CEM) – 機能ブロック

Contrail Enterprise MultiCloud

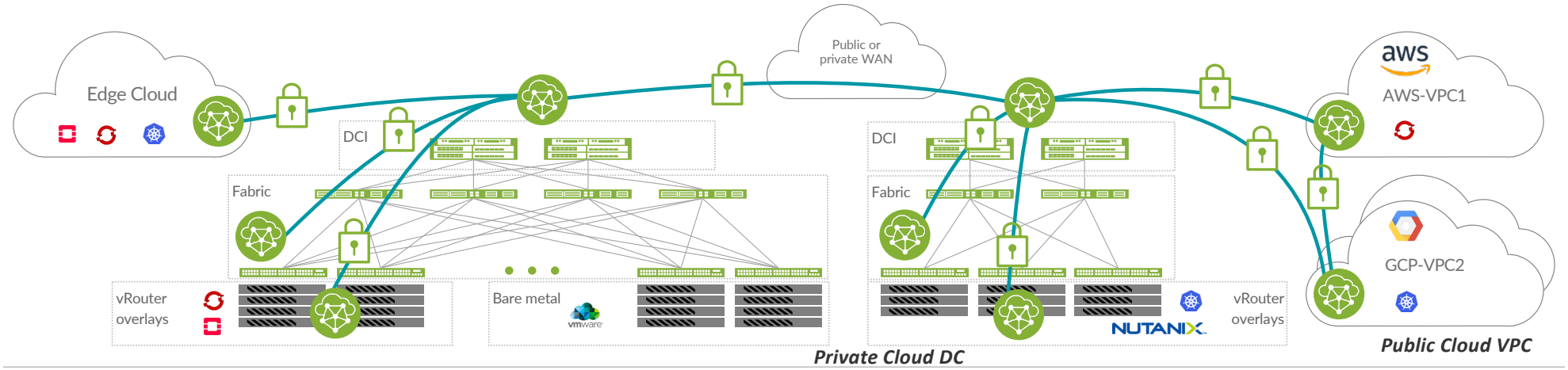
Networking: VM/コンテナ間の仮想ネットワーク制御、サービスチェイン

Security : マルチクラウド統合セキュリティ・ポリシー・オーケストレーション

Insights : テレメトリ、フロー、物理＆仮想可視化、トラブルシュート、ML分析

Fabric : アンダーレイ・スイッチの制御、ZTP、IP Fabric、EVPN-VXLAN、DCI接続

Multicloud : Public クラウド向けとの接続、暗号化

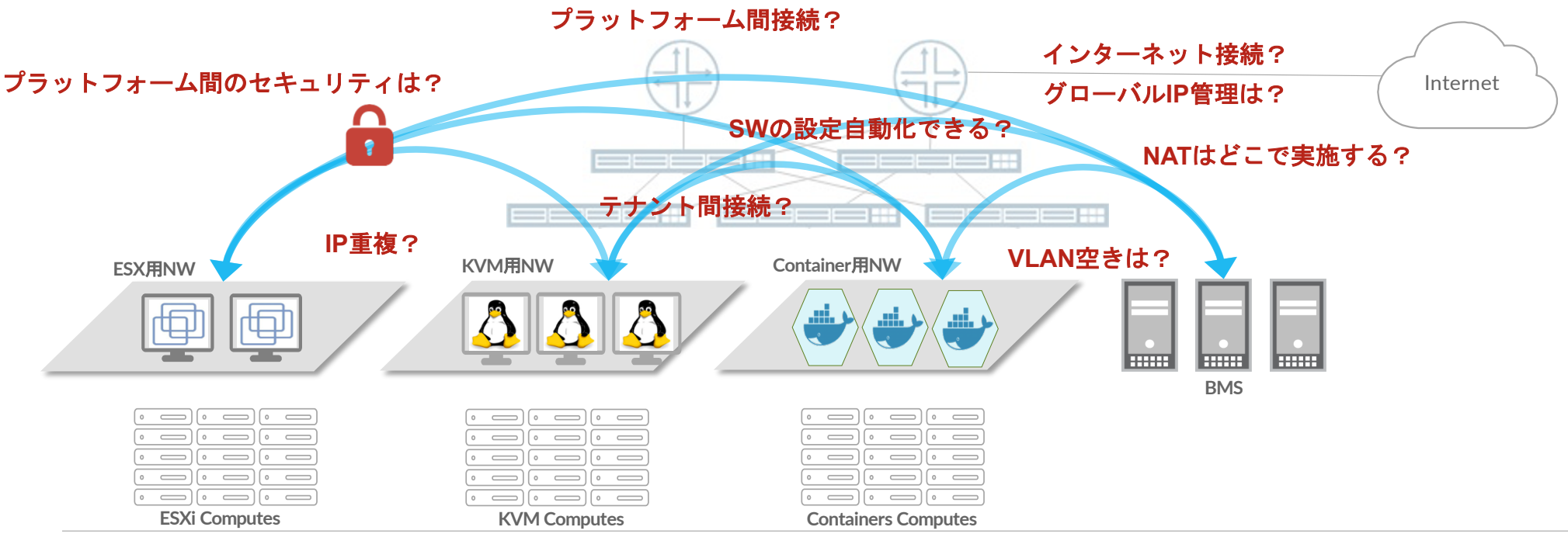


CEM概要

Contrail Networking

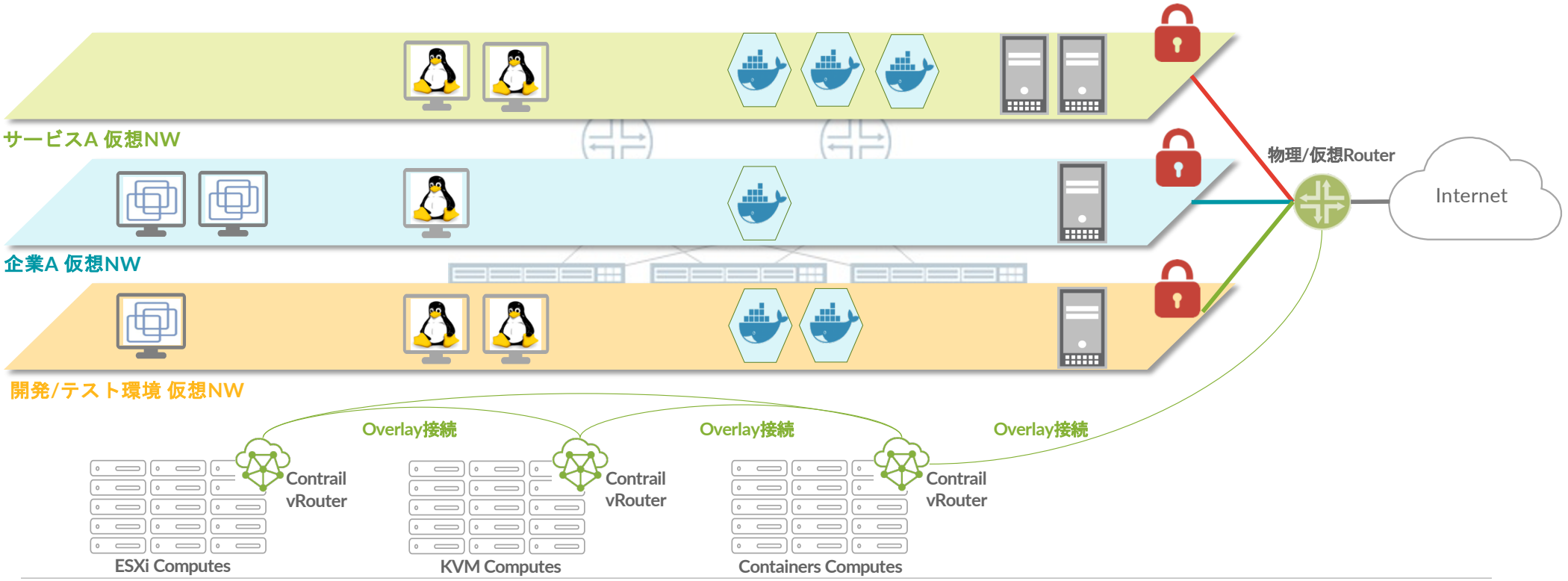
ネットワーク仮想化の課題

- 多様化するアプリケーションニーズに答えるため、VirtualMachine, Container, Baremetal Serverなど、プラットフォームを自由に選択できるインフラ基盤が求められる中、ネットワークはサイロ化されたまま



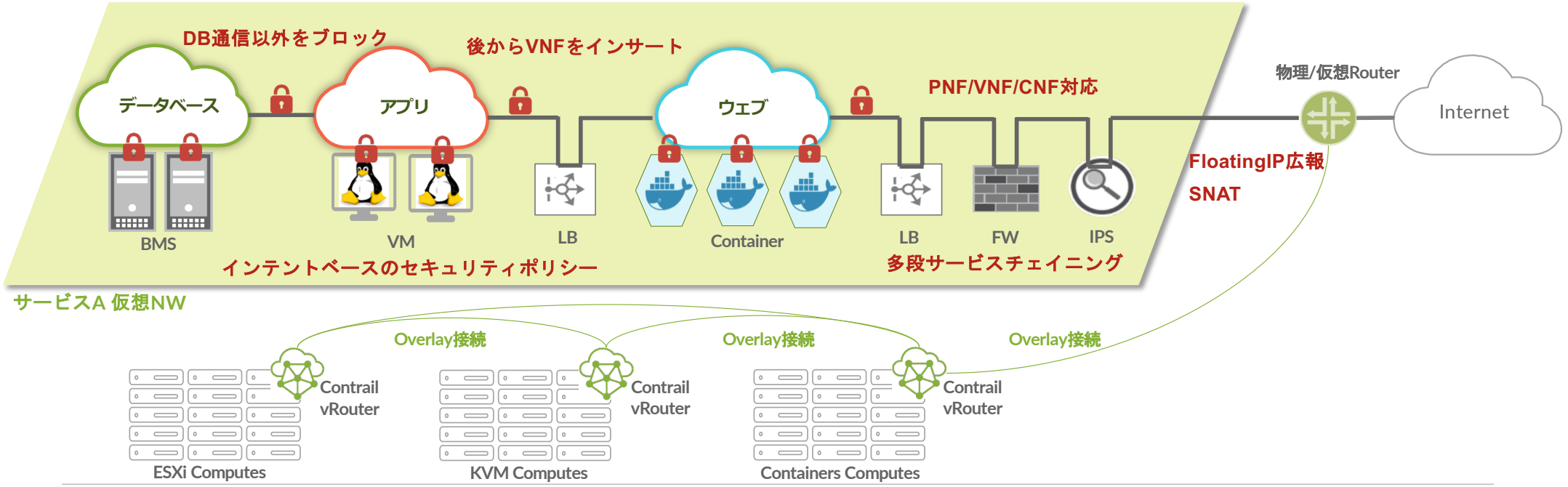
CONTRAILの仮想ネットワーク

- 異種プラットフォームを跨り、サービスや用途に応じて仮想ネットワークをスライシング
- 異種プラットフォームでも共通のセキュリティポリシー、IP管理、インターネット接続の一元管理



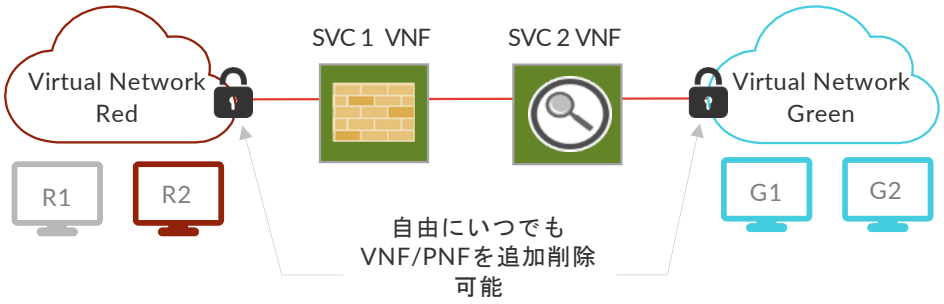
CONTRAILの仮想ネットワーク

- Contrail vRouterの分散仮想FW機能によるプラットフォームを跨った共通のセキュリティポリシー
- IPベースのフィルタリングではなく、サービスにタグ付けしたインテントベースのセキュリティポリシー

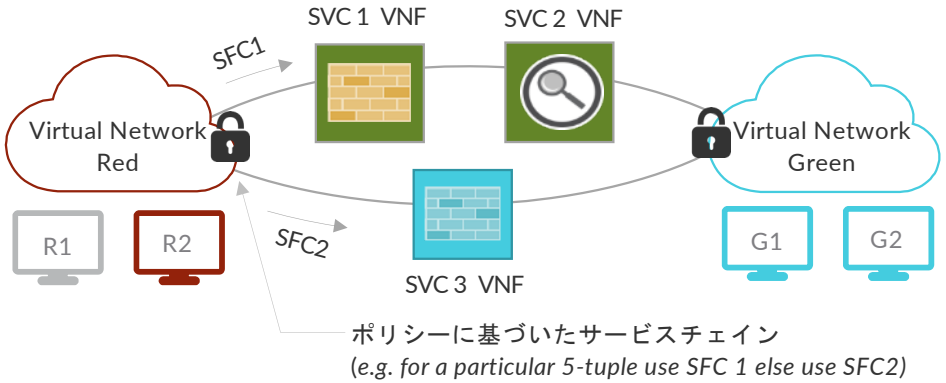


多様なサービスチェイニング

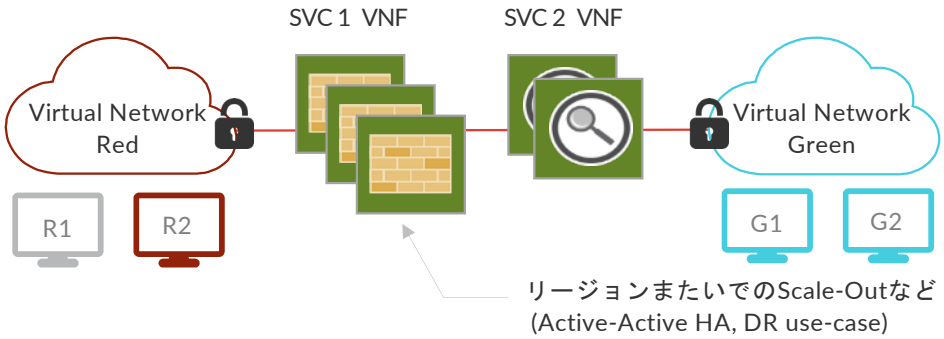
多段サービスチェニング (VNF, PNF)



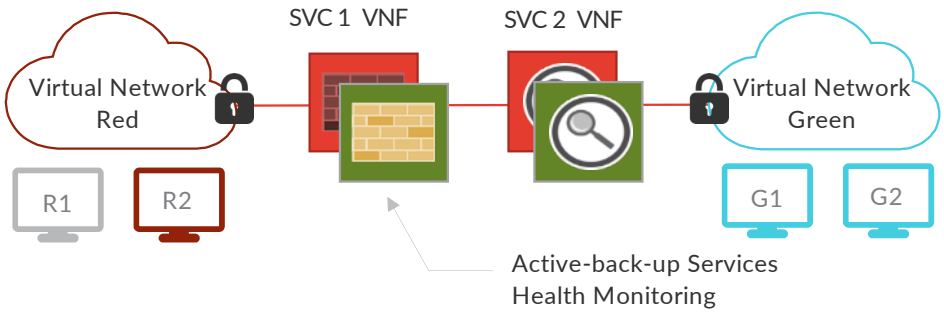
ポリシーベース・サービスチェイニング



スケール・アウト／スケール・イン (Active-Active HA)



アクティブ・スタンバイ

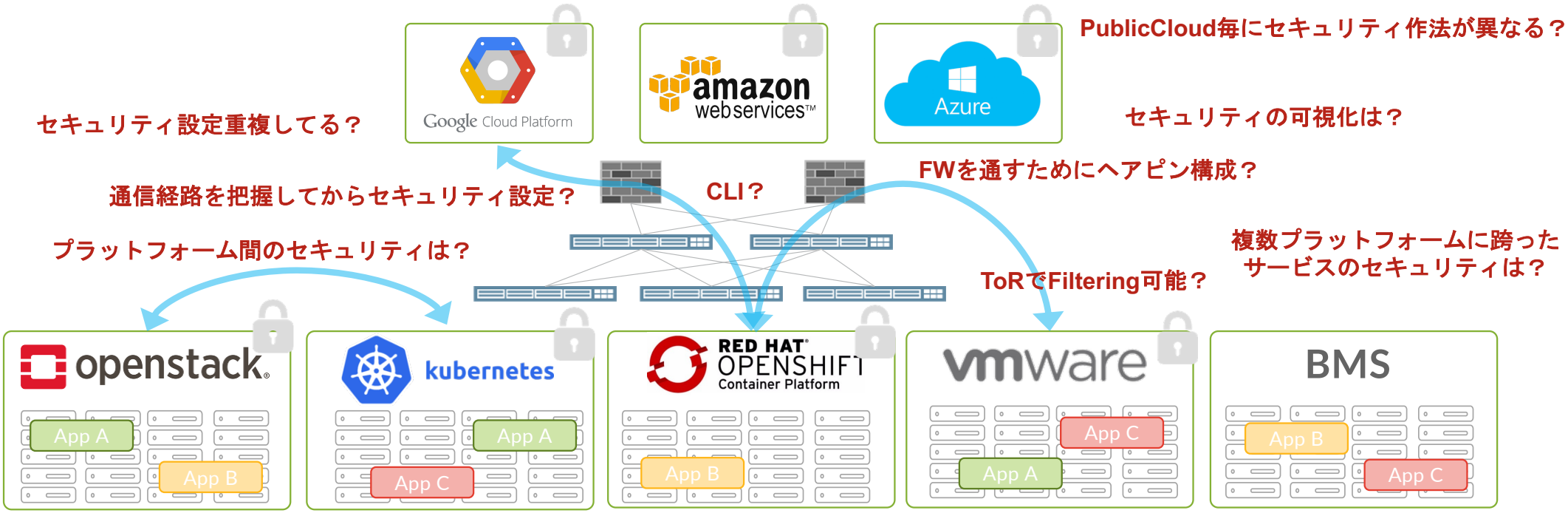


CEM概要

Contrail Security

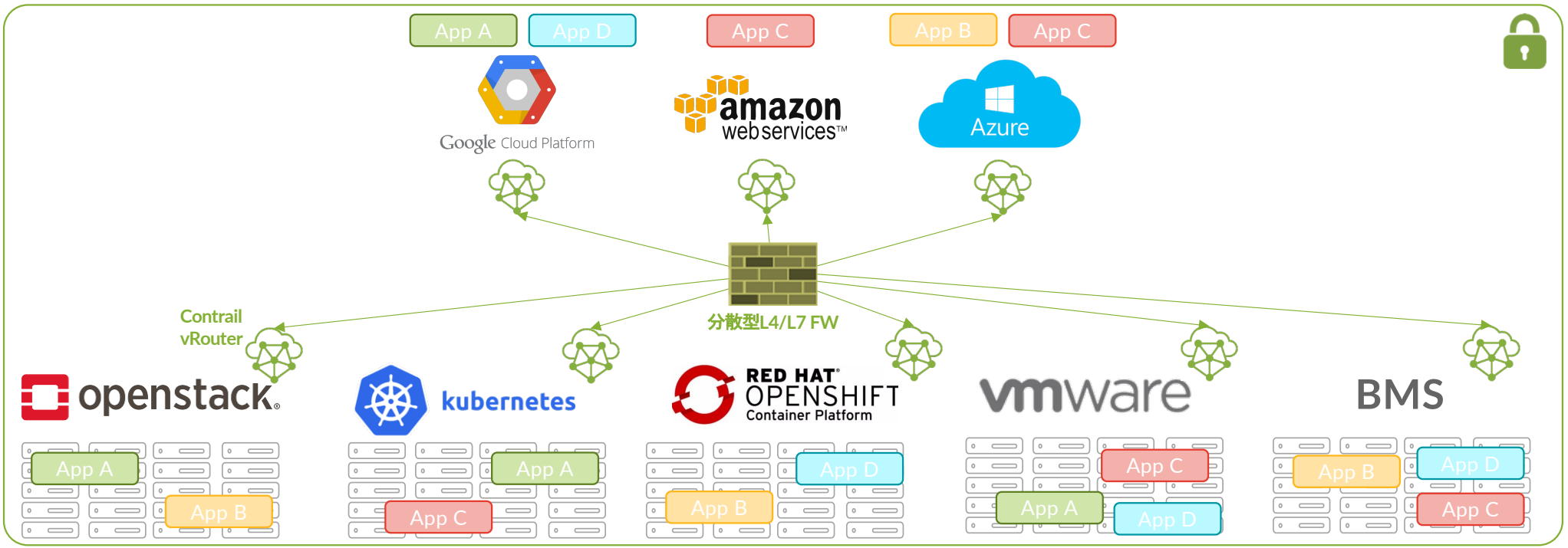
仮想ネットワークのセキュリティ課題

- セキュリティはプラットフォームやクラウド単位で実施されており、一貫したセキュリティが担保されていない
- アプリへの動的なIP付与が一般化され、IPベースのセキュリティ設定は困難に
- Internetや外部ネットワークとの境界に設置するFWだけでは脅威を防ぎきれない



CONTRAILの分散型FW

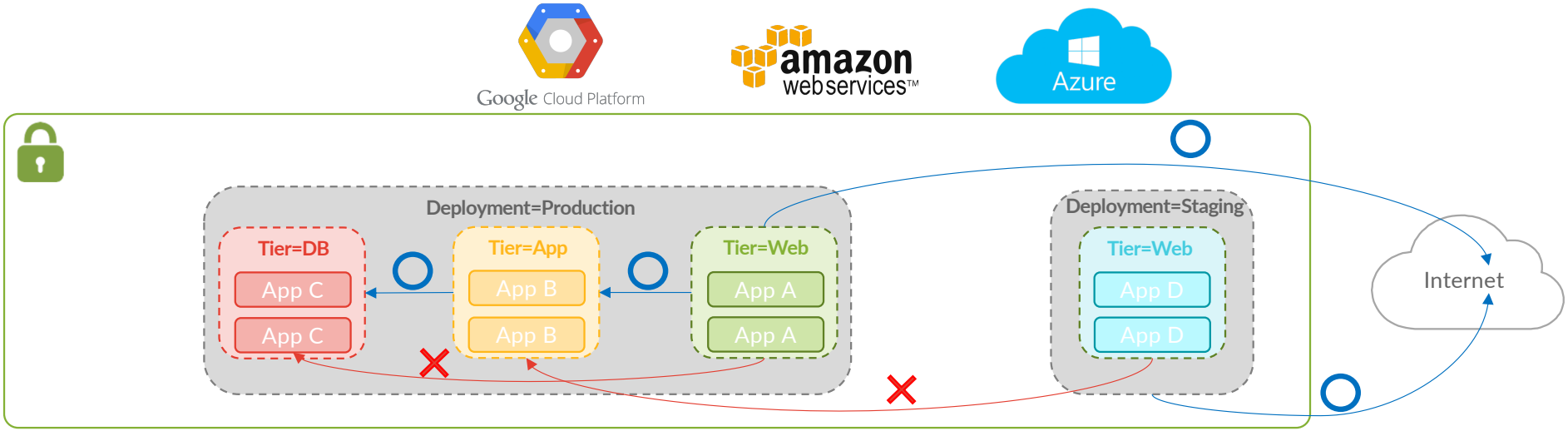
- 各プラットフォームにデプロイされたContrail vRouter Agentが分散型L4/L7 FWとして稼働し、仮想NW全体でセキュリティポリシーを担保
- 分散型FWによりロケーション、プラットフォームに捉われずにセキュリティ設定可能



CONTRAIL SECURITY

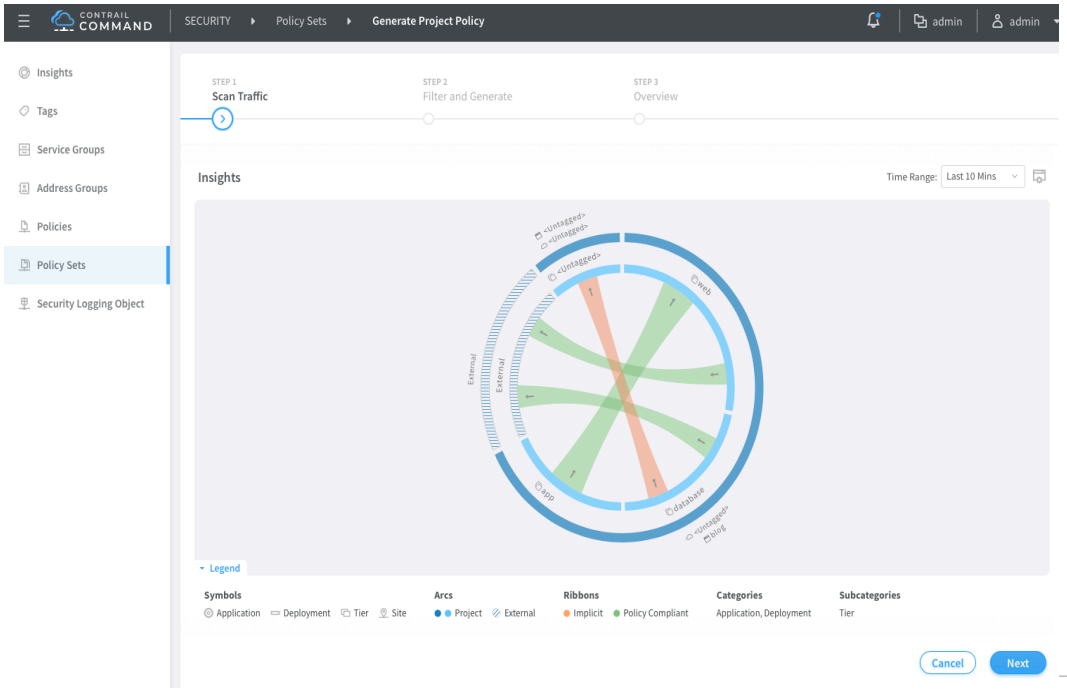
- 分散配置されていたアプリケーションを用途毎にグループ化しタグ付け
- IPベースではなく、インテントベースのフィルタリング
- セキュリティポリシーの一括適用

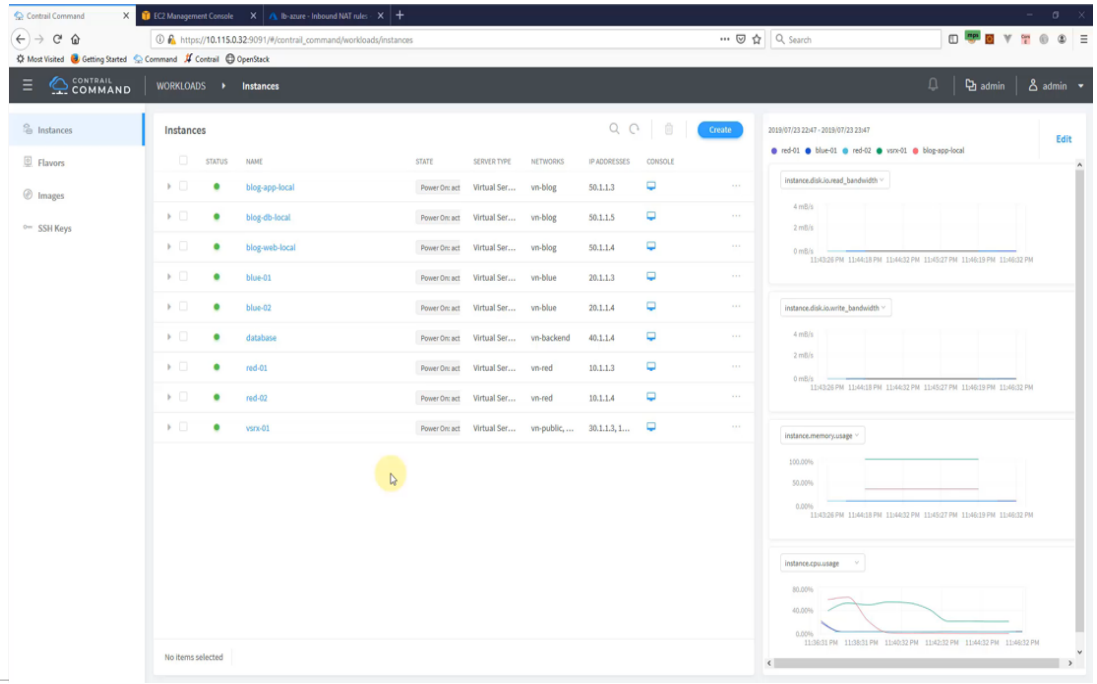
Allow TCP 3036 tier=app > tier=db match Deployment=Production
Allow TCP 80 tier=web > tier=app match Deployment=Production
Deny TCP 80 tier=web > tier=app match Deployment=Staging



CONTRAIL SECURITY GUI

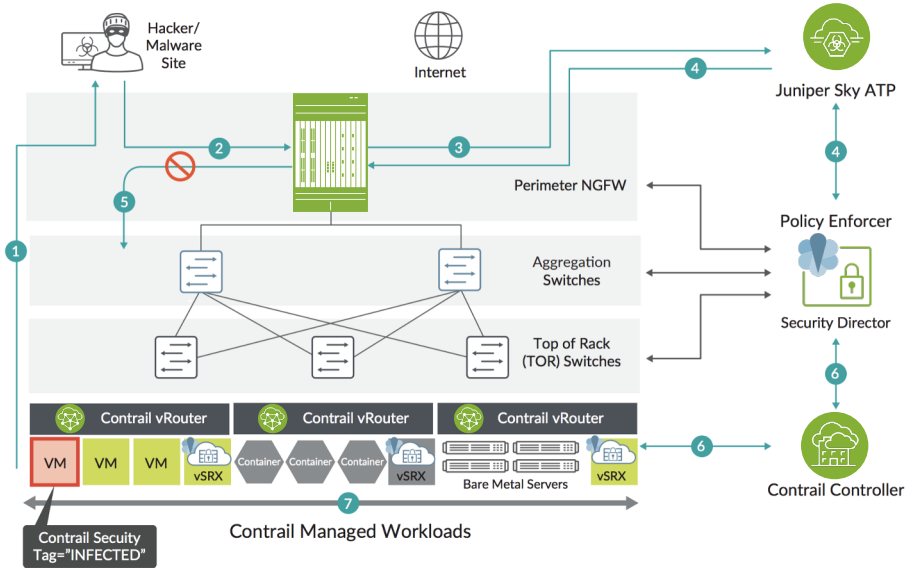
- トラヒックフローの見える化
- タグ付けされたアプリのトラヒックをモニター分析し、必要なセキュリティポシリーを設定可能
- セキュリティポリシー毎のトラヒック流量把握
- 容易なセキュリティポリシーの有効・無効化





脅威インテリジェンスとCONTRAILによる自動隔離連携

Contrail + Security Director Policy Enforcer + ATP



マイクロセグメンテーションを超えて

1. VMがマルウエアファイルのダウンロードを試みている、もしくはC&Cサーベへの接続を試みようとしている
2. vSRXでスキャンを実行
3. vSRXからATPもしくはSkyATPへファイルを送信
4. ATPがマルウエアかどうかを判断しポリシーエンフォーサへ通知
5. vSRX、Switchにて自動で隔離を実行

Integration with Contrail:

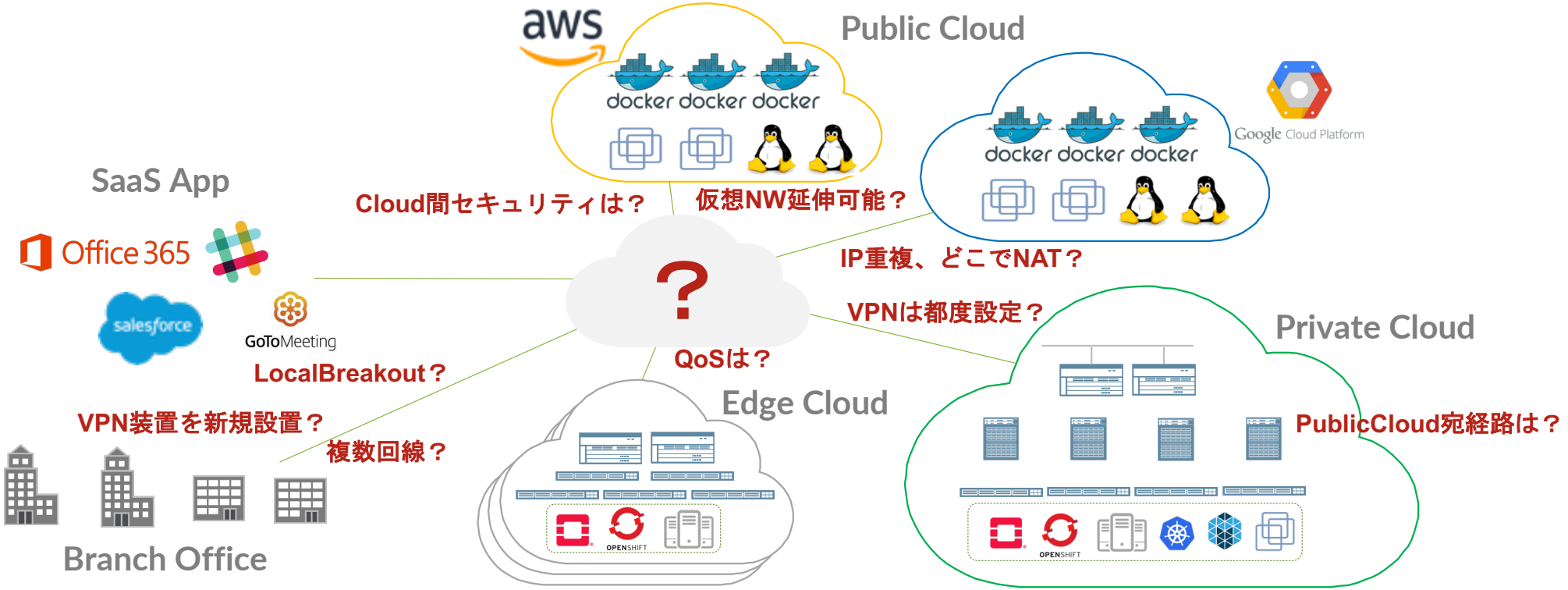
6. ポリシーエンフォーサがContrailへ適切なセキュリティ・タグを付けて隔離Security Groupへ隔離
7. セキュリティ管理者で事前に対応アクション定義や、さらなるアクションを定義

CEM概要

MultiCloud

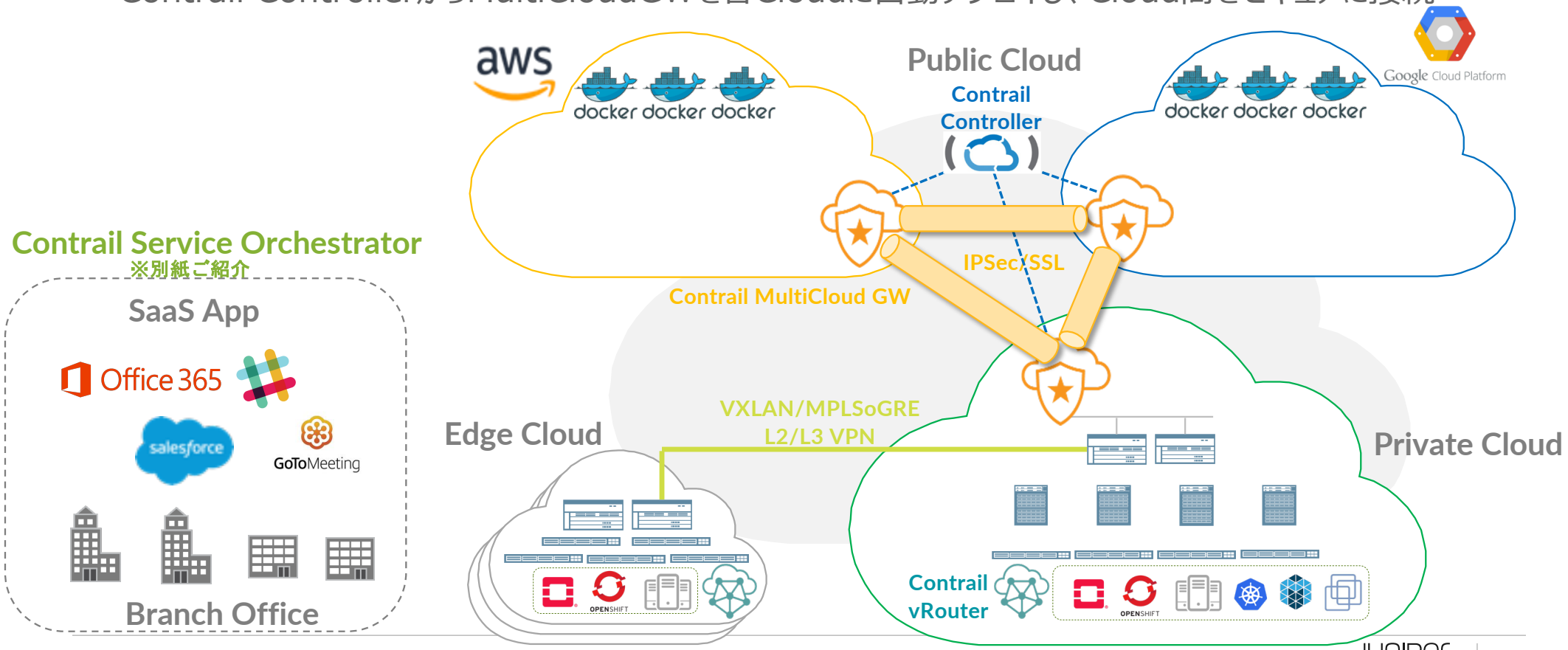
MULTI-CLOUD ≠ MULTIPLE CLOUD

- それぞれのCloudやSaaSの特徴を活かし、用途に応じ複数のCloudを選択する時代へ
- サイロ化されたCloud間をVPN Routerで接続するだけでは不十分



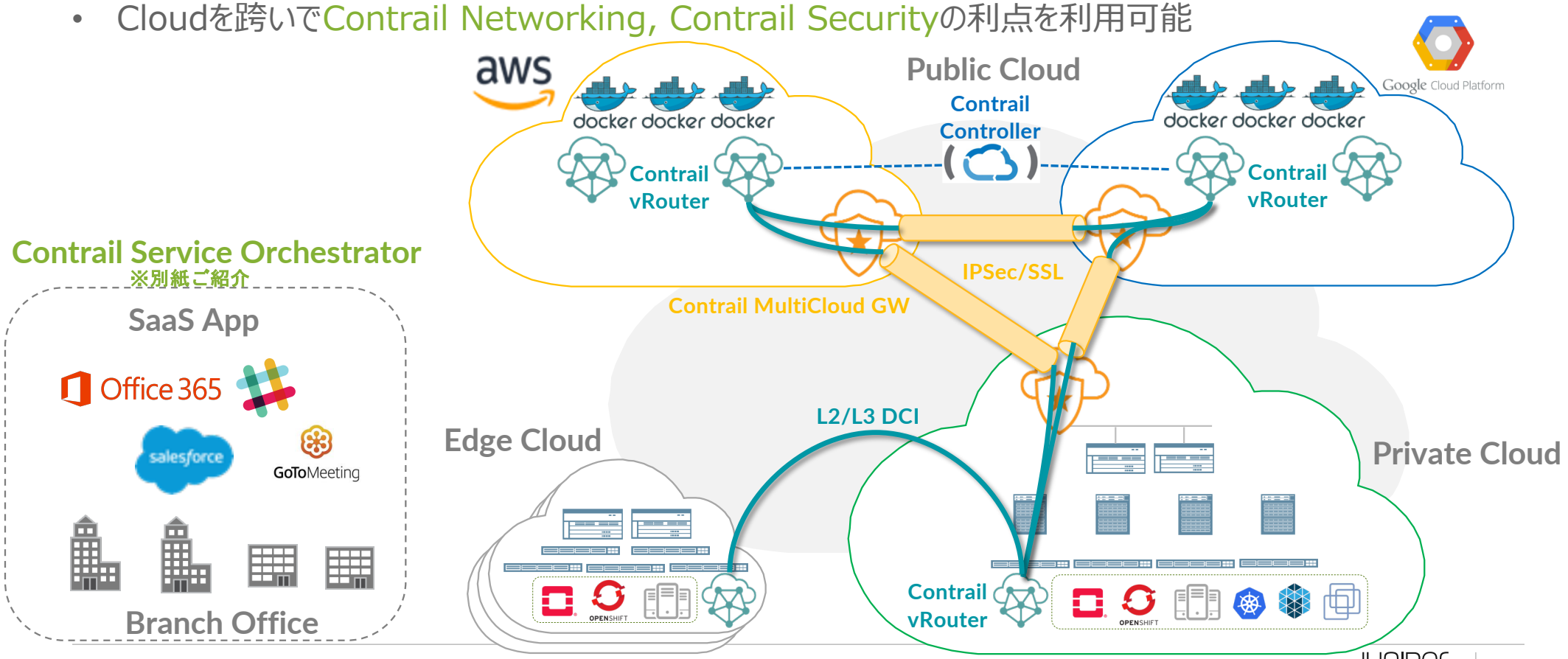
CONTRAIL MULTI-CLOUD

- Contrail ControllerからMultiCloudGWを各Cloudに自動デプロイし、Cloud間をセキュアに接続



CONTRAIL MULTI-CLOUD

- 仮想NWを提供するvRouter間接続はMultCloudGWで作成したTunnel上で接続される
- Cloudを跨いでContrail Networking, Contrail Securityの利点を利用可能

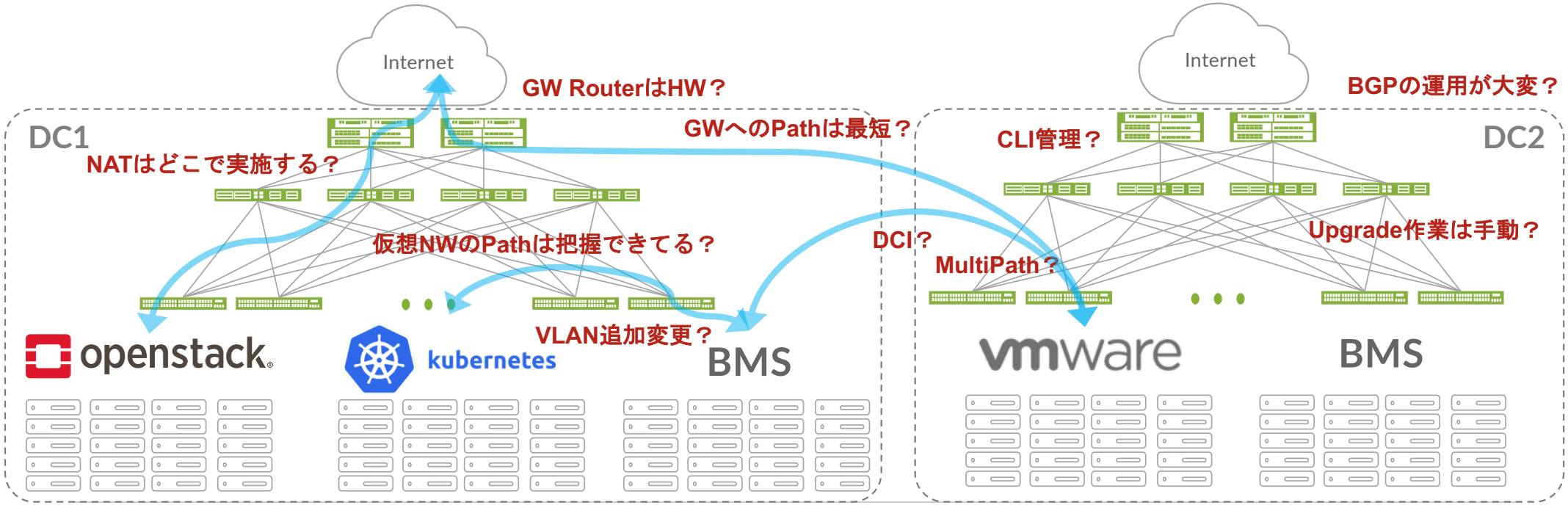


CEM概要

Fabric Automation

DC FABRICの課題

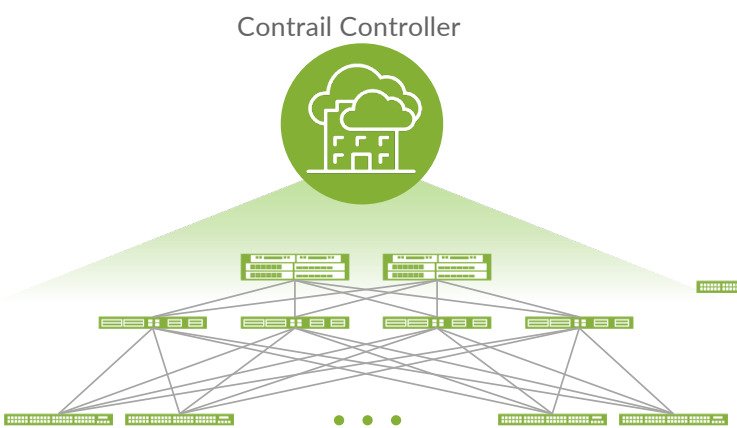
- DC Fabric と各プラットフォームの仮想ネットワークは別々に管理運用されているケースが多く、仮想ネットワークの追加変更に伴い、DC Fabricの変更を余儀なくされる
- 外部アクセスに伴うNATやVLAN追加、経路制御、DC Interconnectなどの課題が残り、仮想ネットワークに最適なDC Fabricにはなっていない



CONTRAIL FABRIC AUTOMATION

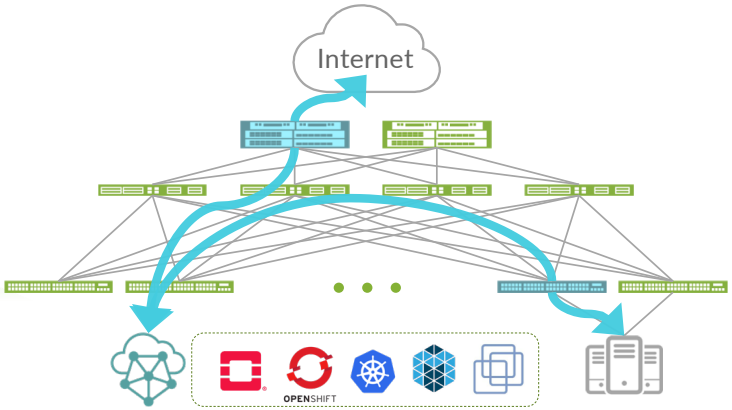
スケーラブルファブリックのライフサイクルを自動化

DAY-0 : セットアップ



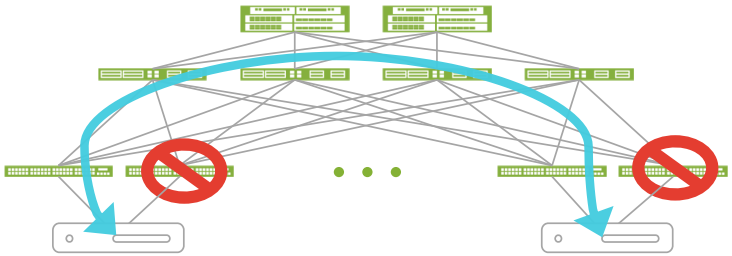
- Contrail Controllerからファブリック全体を管理
- 物理機器はラッキング、ケーブリングのみ実施
- ファブリックの設定は全て自動化
- Multi DC Fabric対応

DAY– 1 : 運用



- 仮想NWを物理機器に自動設定
- BMSを簡単に仮想NWへ接続
- GW Routerで仮想NWを終端し外部接続
- PNFを使用したServiceChaining

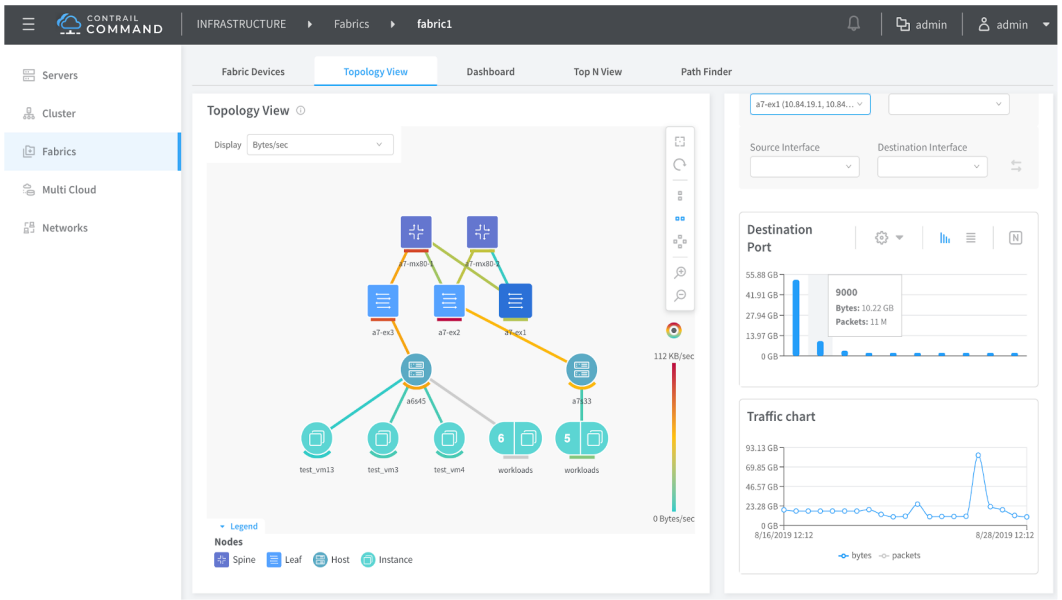
DAY-2 : メンテンス

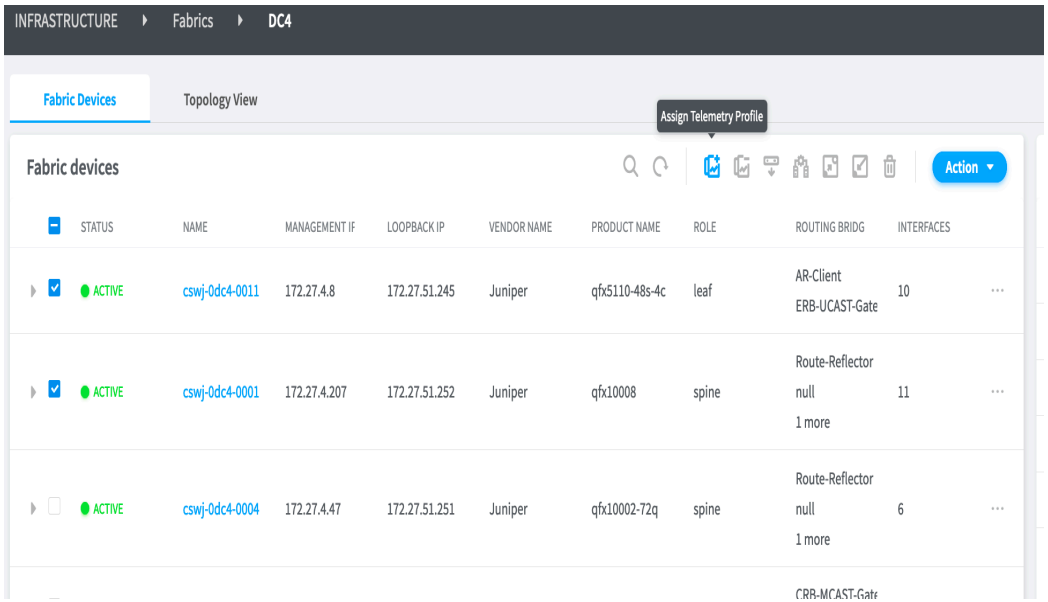


- メンテナンスモードによりトラヒックを自動迂回
- ファブリック全体をロスレスアップグレード
- 容易なスケールアウト
- Configを維持した容易な機器交換

CONTRAIL FABRIC GUI

- Fabric機器の一元管理(Device種別、Version, Serial, Interface, IP, Role, etc)
- Fabric全体のトポロジー表示
- 仮想NW間のPath表示
- テレメトリによるリアルタイムトラヒックモニタリング





CEM概要

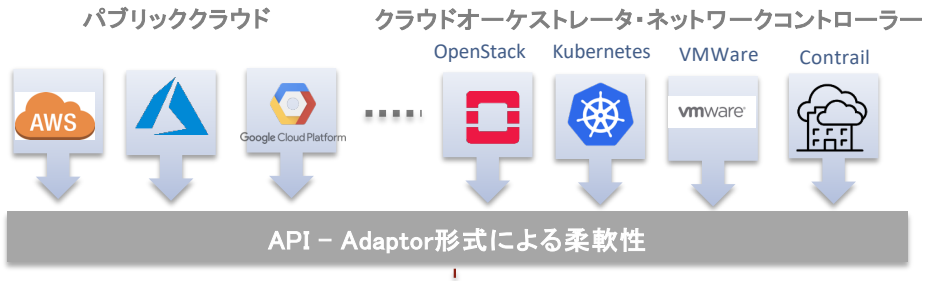
Contrail Insights

(AppFormix)

CONTRAIL INSIGHTS

マルチクラウド環境の統合リアルタイム可視化

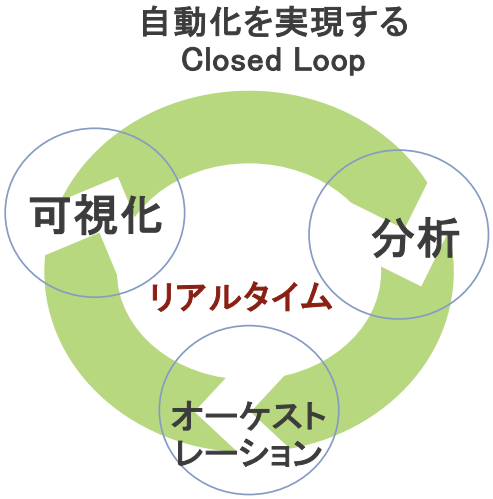
CONTRAIL INSIGHTS



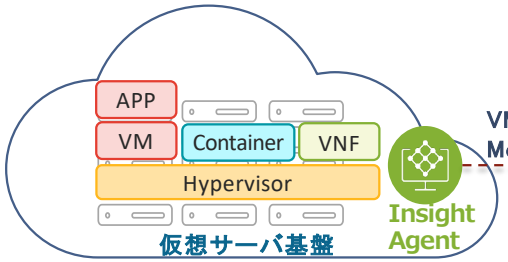
✅ APIを使用した柔軟なマルチクラウド対応

Controllerリソース情報

Contrail Insights



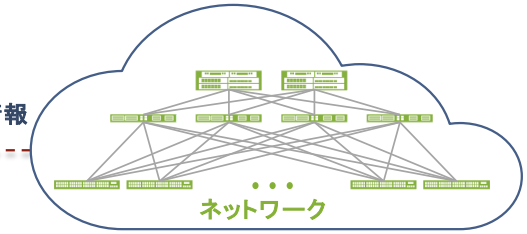
✅ 軽量なAgentによるリソース監視



VM, Project, VNF
Metric情報

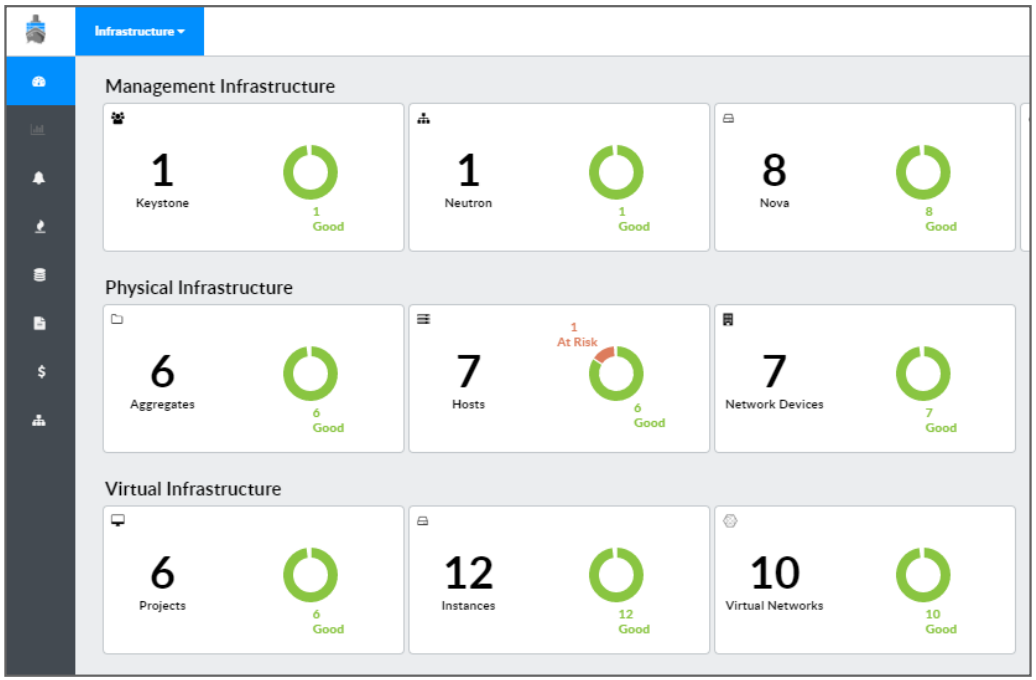
MIB/Telemetry情報

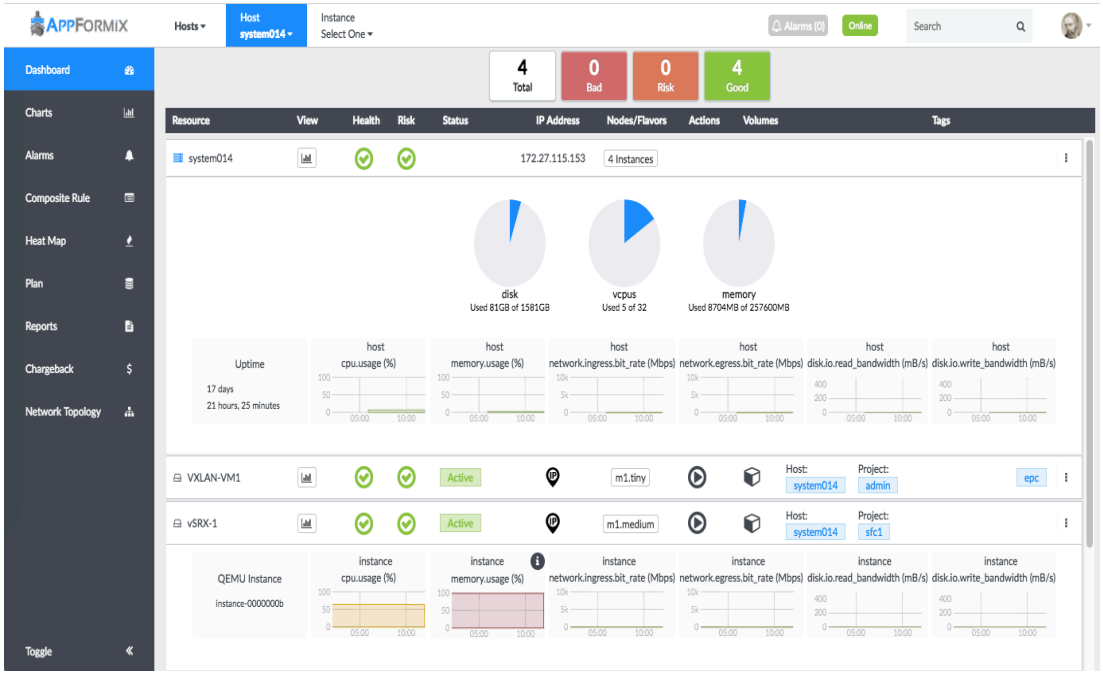
✅ Telemetry/SNMPによるNWの可視化



クラウド、アプリケーション　モニタリング

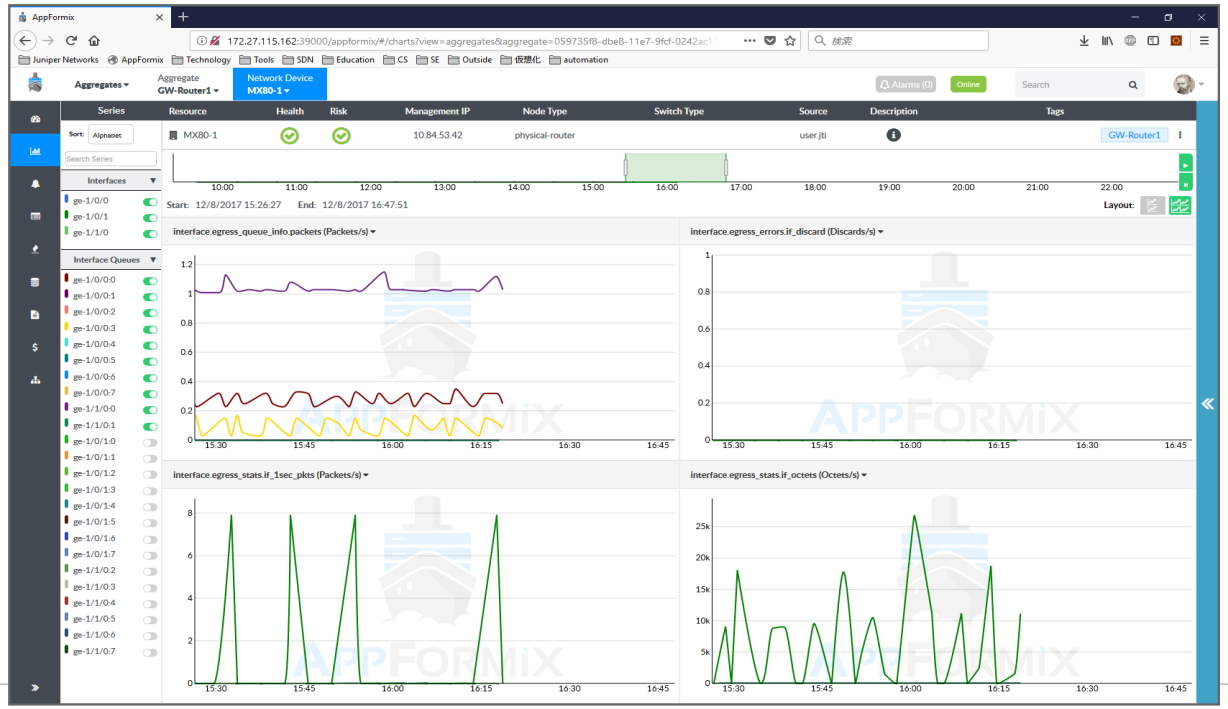
- 物理デバイス、仮想デバイス問わずプロセスのヘルスモニタリング
- CPU, Memory, Network, Diskなどのリソース使用状況把握とキャパシティプランニング





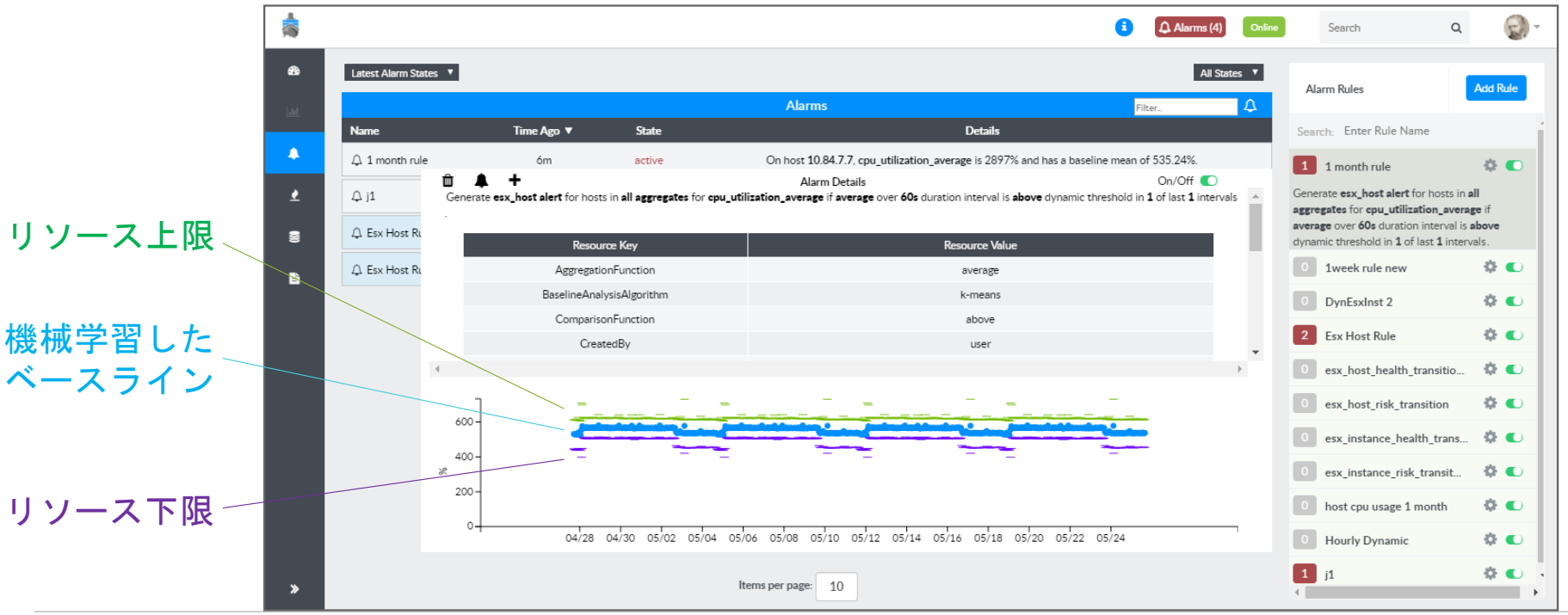
ネットワークモニタリング

- テレメトリによるリアルタイムネットワークモニタリング
- デバイス、インターフェース、Queue単位のモニタリングやマイクロバーストの把握が可能

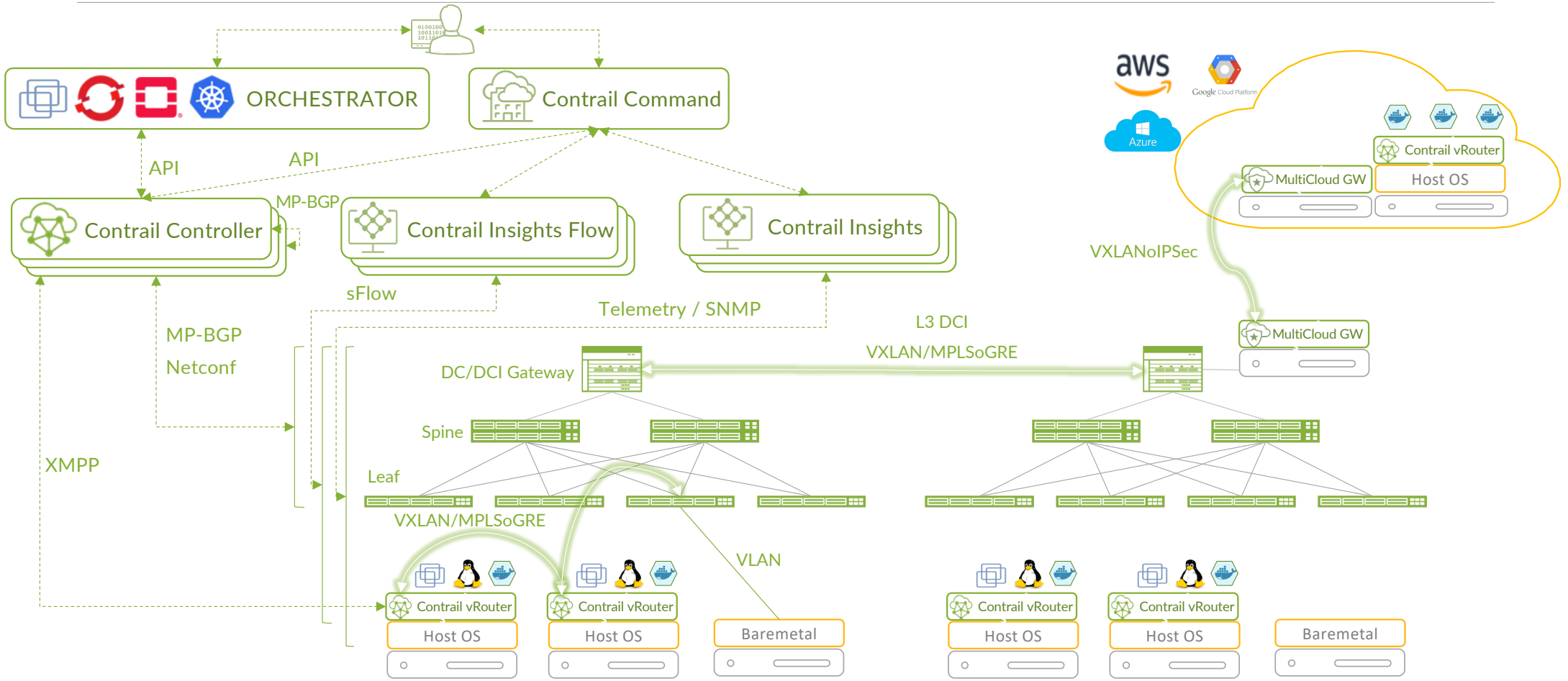


機械学習による予兆分析とダイナミックアラーム

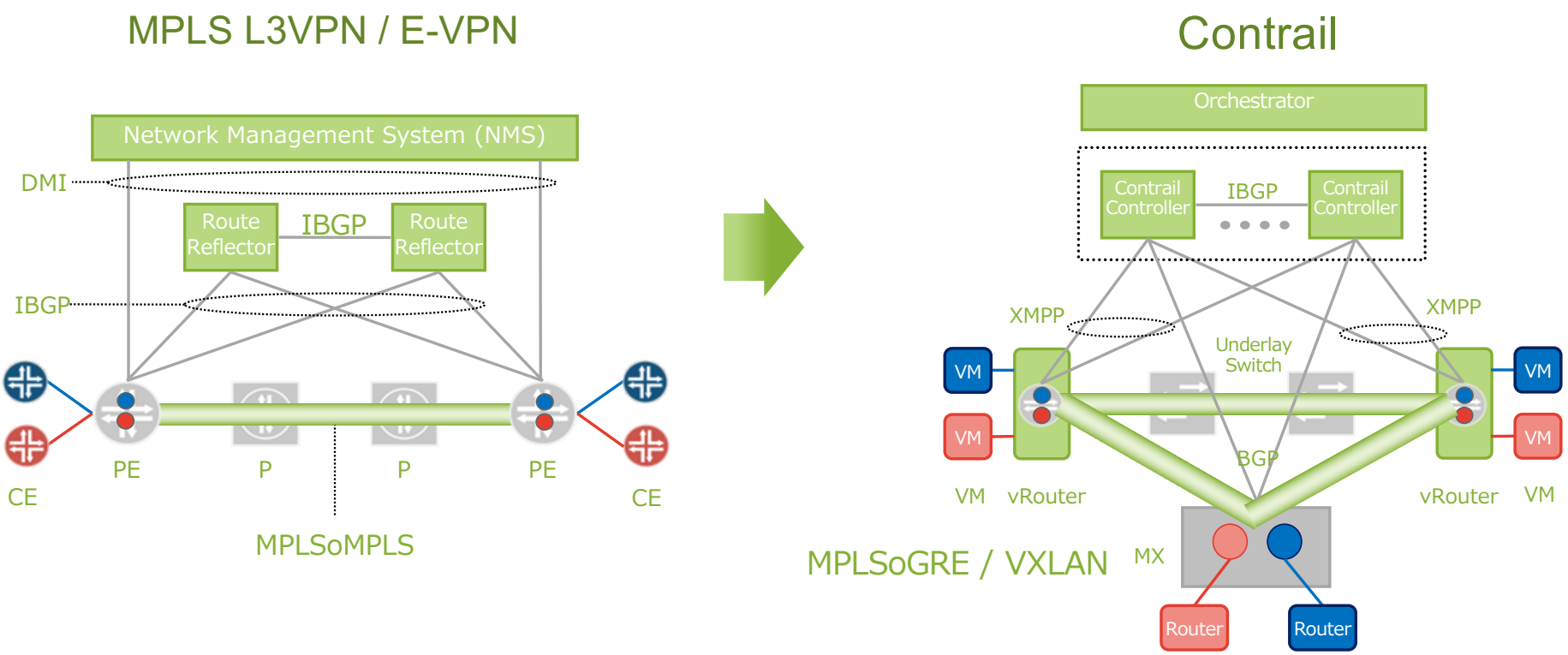
- VLAN, Interface Queueのドロップやバーストトラヒックの傾向分析
- メモリーリーク、CPUバースト、Disk I/O レスポンスタイム悪化など傾向分析
- 分析結果から動的にベースラインを作成し、アラーム設定



CEMアーキテクチャ

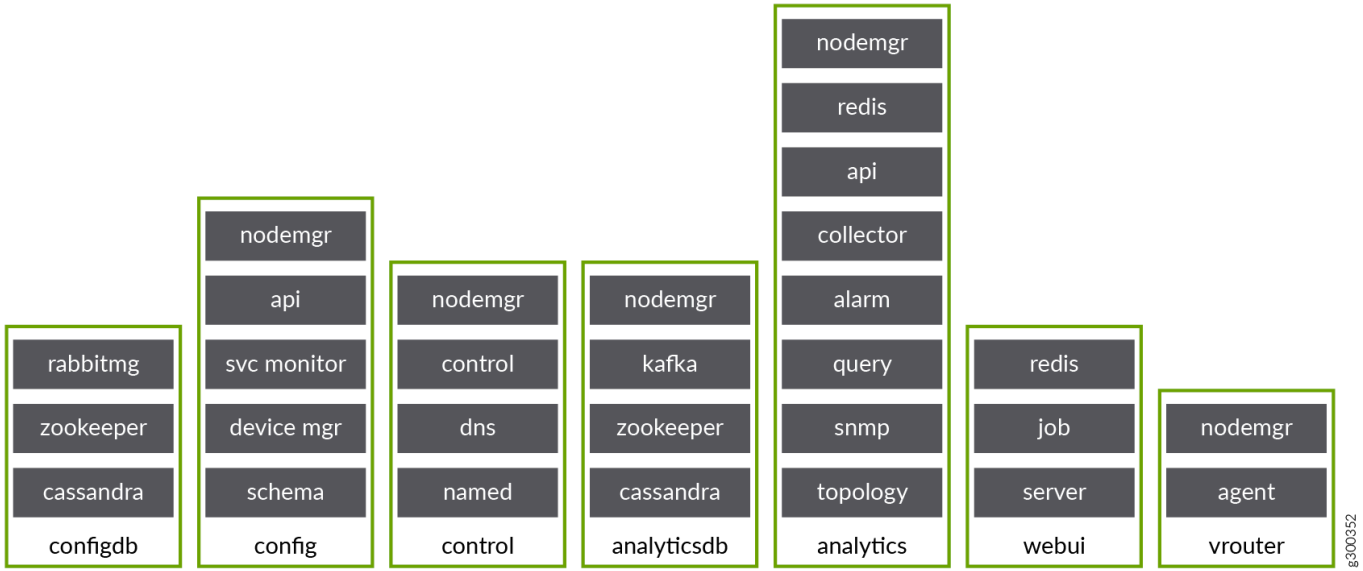


MPLS/L3VPN ベースアーキテクチャ

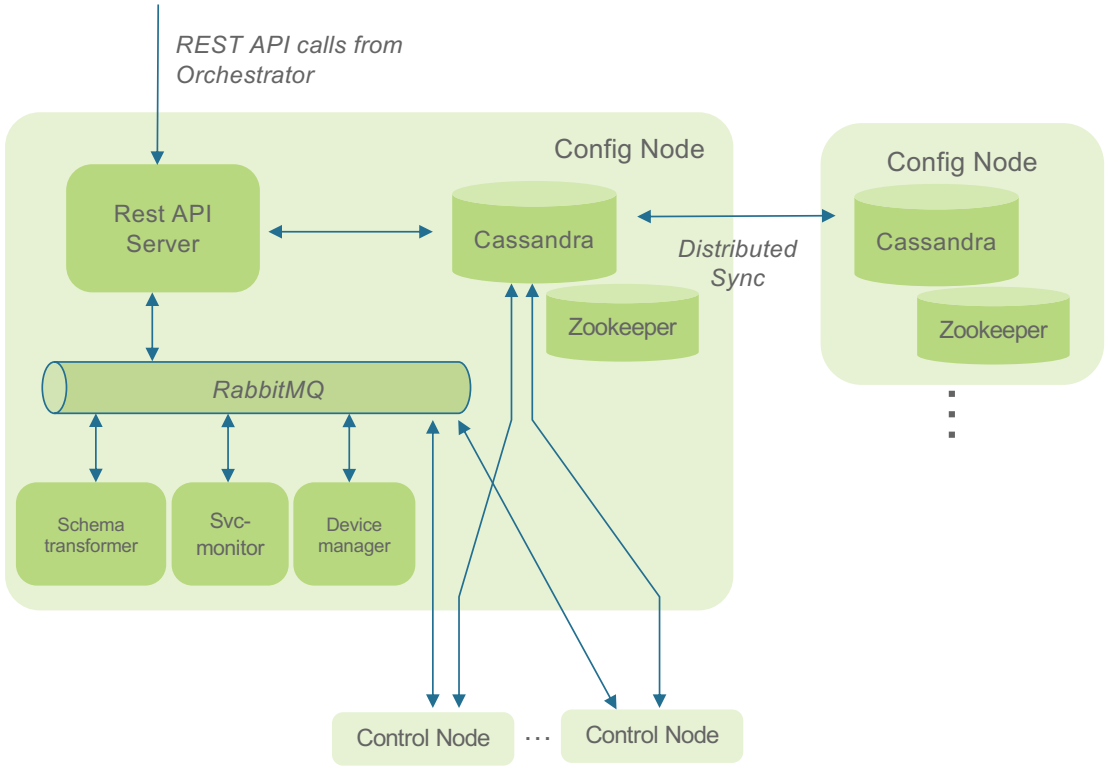


Contrail Controllerは複数のMicroserviceで構成されている

- webui：Contrail Controller GUIを提供(Contrail Commandとは別)
- control：vRouterとのXMPP接続、Control/Router間のBGP接続を行う
- config：API Requestを受け、仮想NWやエンドポイントの追加を行う
- configdb：config情報のDB
- analytics：ネットワーク要素全体のデータ収集、保存、分析、関連付けを行う
- analyticsdb：analytics情報のDB
- vrouter：Compute Node上で動作するAgentとなり、VM/ContainerのL2/L3接続を行う

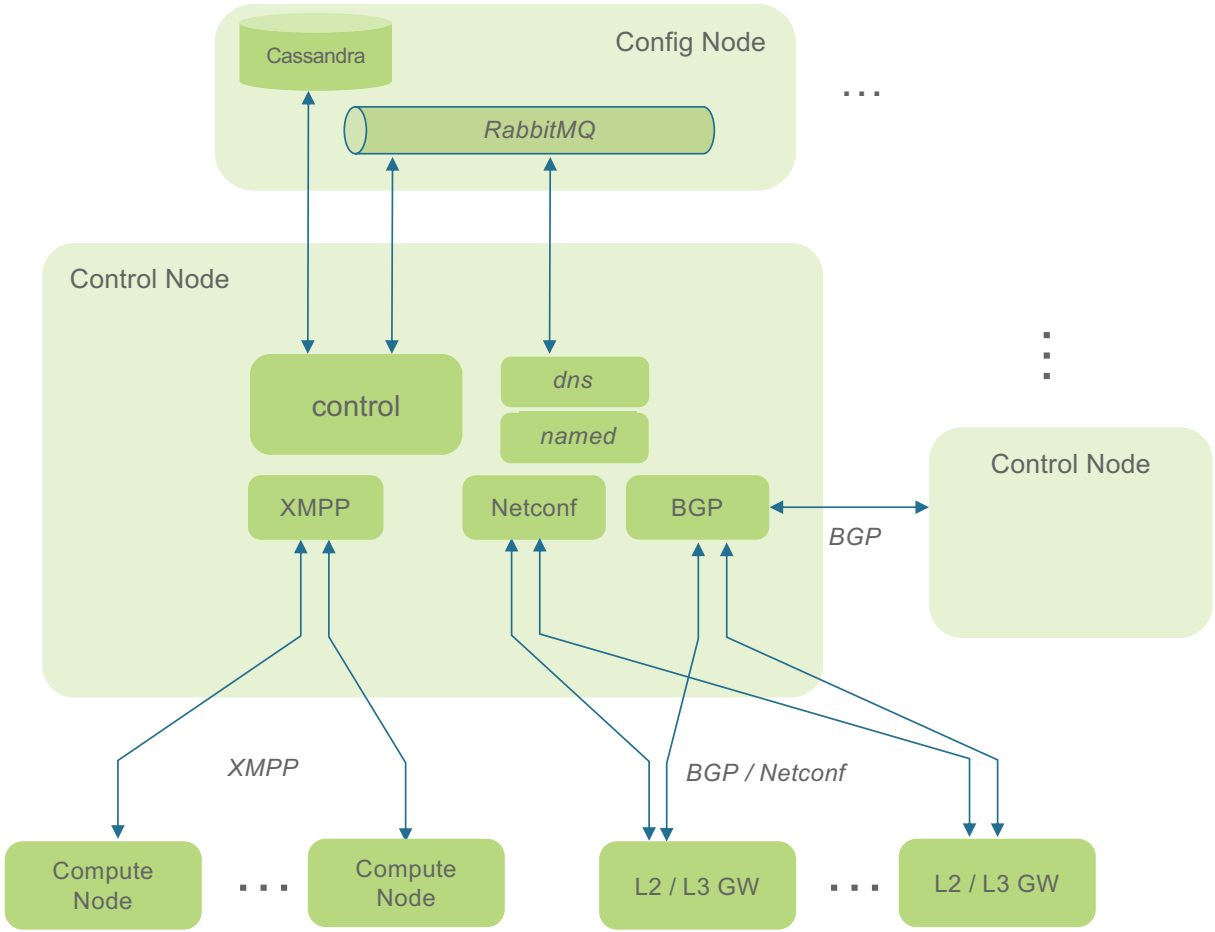


CONTRAIL CONTROLLER - CONFIG NODE 内部アーキテクチャ



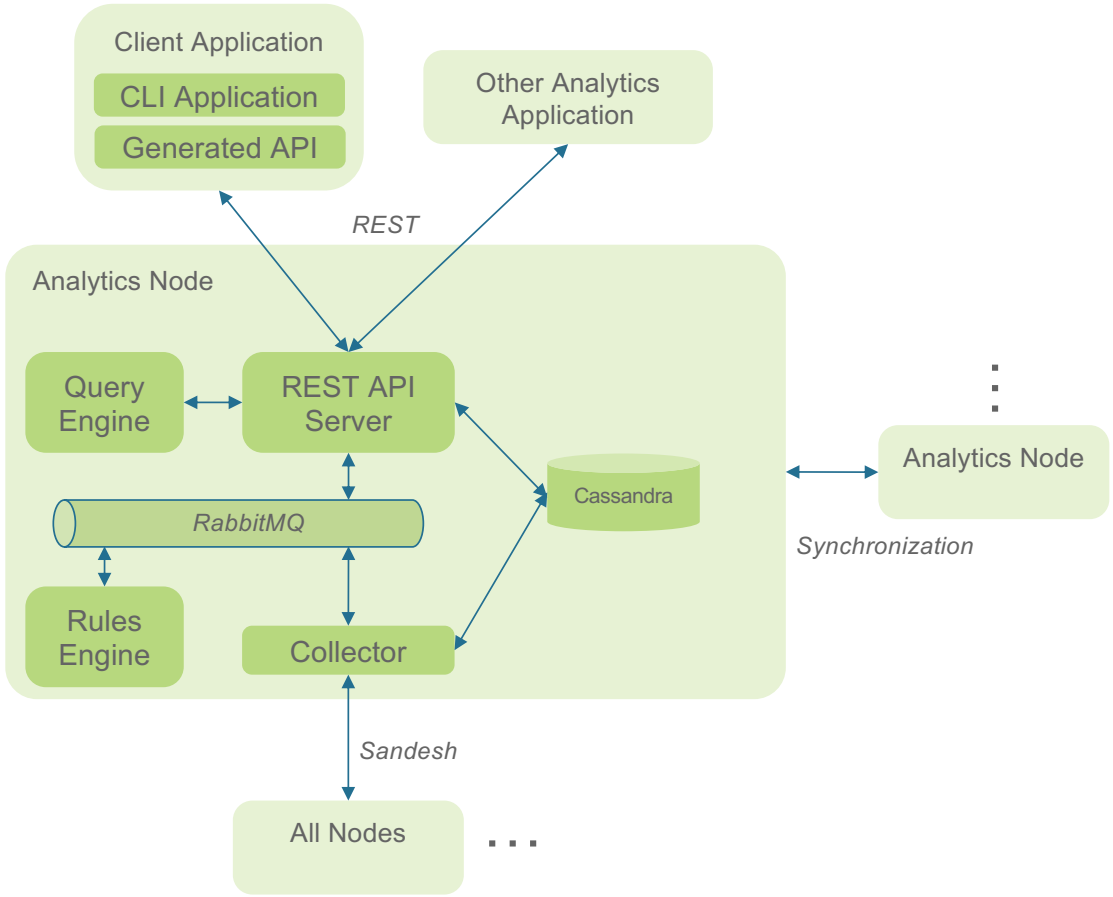
- Rest API Serverはオーケストレーションシステムまたは他のアプリケーションとのノースバンドインターヘースを提供。本インターフェースはハイレベルデータモデルを使用し、コンフィグ情報のイントールを行う。またDBへのデータストア、ロードを行う
- RabbitMQは内部コンポーネント間のメッセージバス
- Cassandra DBはコンフィグ情報の保持
- Schema Transformerはメッセージバスを介しハイレベルデータモデルからローレベルデータモデルへの変換を行う
- Zookeeperは一意のオブジェクト識別子の割り当てに使用され、マスターシップ選出を行う

CONTRAIL CONTROLLER - CONTROL NODE 内部アーキテクチャ



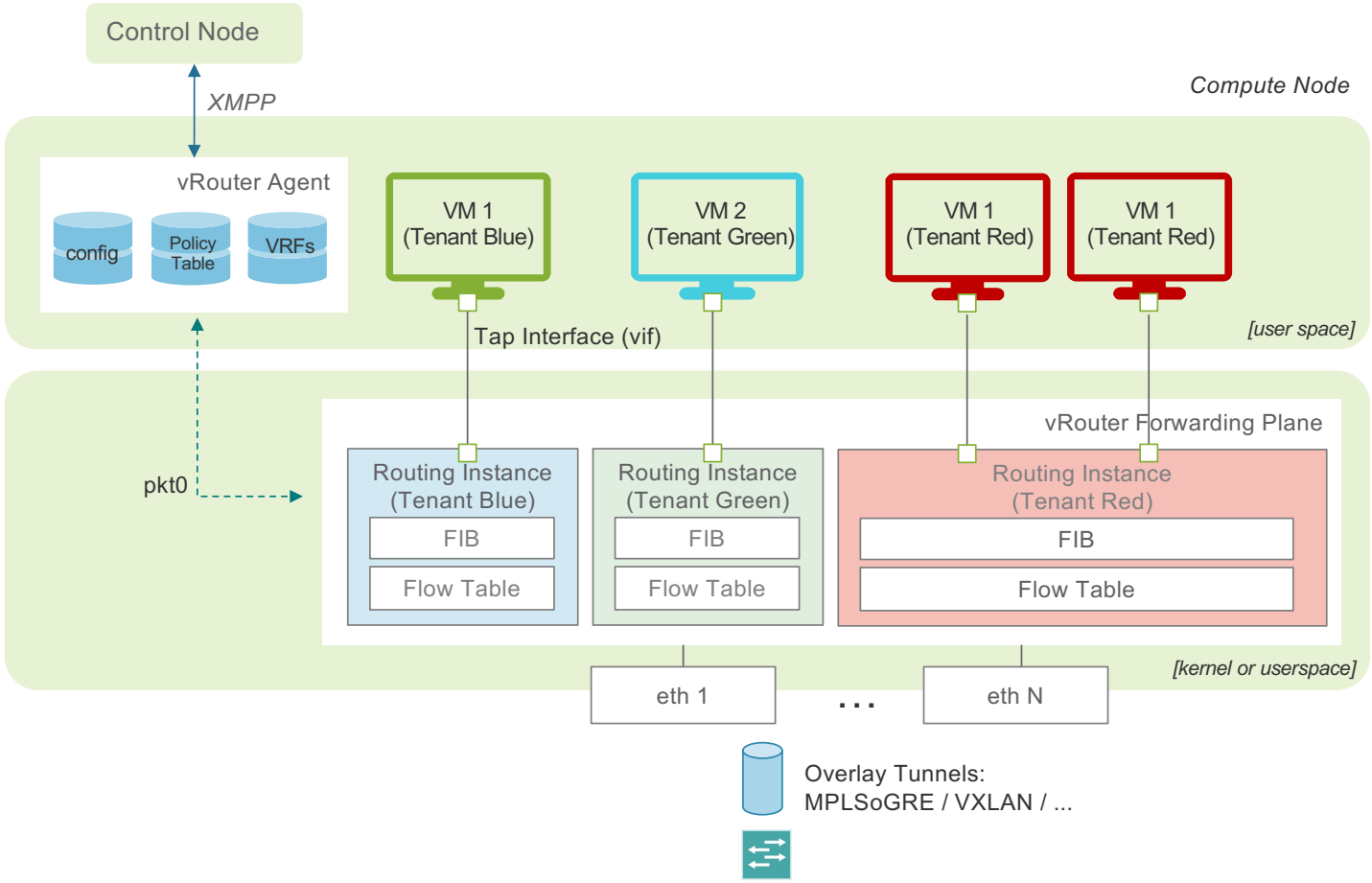
- 全てのControl NodeはActive / Active
- 各vRouterは複数Control NodeにXMPP接続を行う
- 各Control Nodeは複数のConfig Nodeに接続を行う
- ControlはDBから初期コンフィグを読み取り、メッセージバスをリッスンしControl-APIからの通知を受け取る
- Control NodeはRouter/SwitchとBGP接続を行う
- Control Node間でBGP接続を行う

CONTRAIL CONTROLLER - ANALYTICS NODE 内部アーキテクチャ



- vRouterやControl Nodeからの情報をAnalytics Nodesが収集
- 全てのコンポーネントはAnalyticsデータをSandeshでカプセル化してCollectorに送信

CONTRAIL VROUTER 内部アーキテクチャ



- vRouterはLinux BridgeまたはOVSの置換え
- vRouterはbridging(EVPN), routing(L3VPN)を担う
- vRouter Agentは最大2つのControl NodeとXMPP接続を行う
- vRouterはNetwork Service(Security Policy, NAT, Multicast, Mirroring, Load Balancing)を行う
- RouteはPolicyに従いVRFへ自動的にLeakされる
- VM Multi Interfaceをサポート
- Compute/Fabric間のMulti Interfaceをサポート

COMPUTE NODE インターフェース

eth0 / bond0

- Underlay接続用の物理インターフェース(Control/Data Interface)

vhost0

- HostOSへのL3論理インターフェース
- Contrail vRouterインストール時にeth0/bond0インターフェースのIP設定がvhost0インターフェースへ書き換えられる
- HostOSがvhost0インターフェイスでパケットを送信すると、それらはvrouterモジュールで受信され、vRouterがパケットをルーティング。
- vrouterがHostOSにパケットを送信する必要がある場合、vhost0インターフェイスを介して送信。

pkt0

- vRouter Kernel ModuleとvRouter Agent間のインターフェース
- vRouterはpkt0インターフェースでフロー設定などの制御処理を必要とするパケットを受信

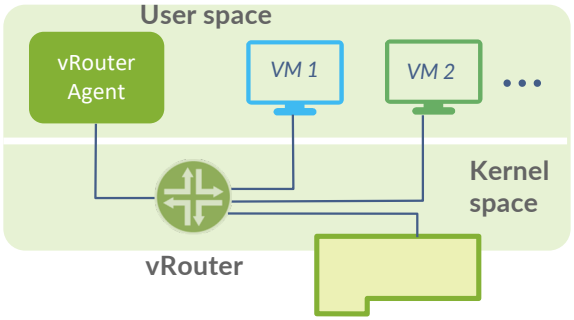
tap-xyz

- ComputeNode内のVM用インターフェス

pkt1/pkt2/pkt3 (vrouter kernel module only, not used with DPDK)

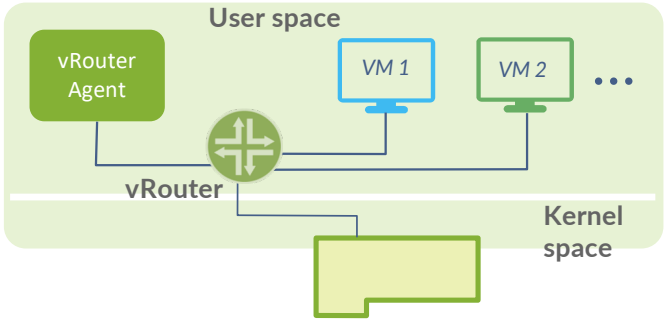
- 受信したパケットを複数のCPUコアに配信し、GRO（TCPトラフィックの受信オフロード）を実行するために使用
  - pkt1/pkt3 – パケットをデキャップ後、VMへ送信する前にSmall TCPセグメントをアグリゲート。pkt1はL3パケット用、pkg3はL2パケット用。また、Linux RPS（Receive Packet Steering）を使用して、内部ヘッダーに基づいてこのような集約セグメントを他のCPUコアに配布
  - pkt2 - Linux RPS（Receive Packet Steering）を使用して、外部ヘッダーに基づいて他のCPUコアに配布

KERNEL VROUTER



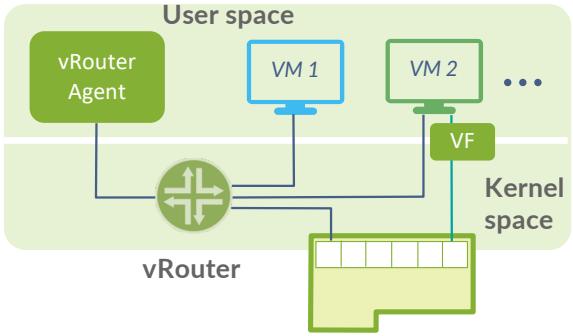
- 通常のモード
- vRouterはKVM Kernelで稼働しIPTables, OVS機能を担う

DPDK VROUTER



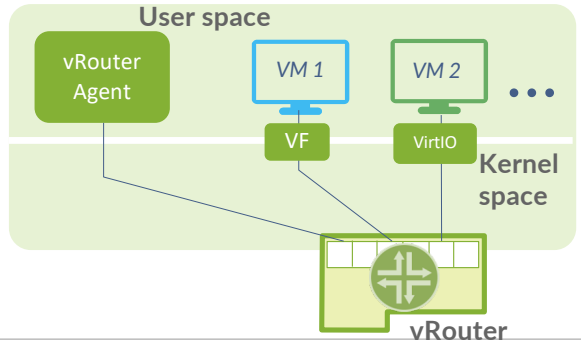
- vRouterはUserSpaceで稼働し、Fast Path Packet I/OのためDPDKを使用
- VM側でDPDK有効化が必要

SR-IOV - VROUTER



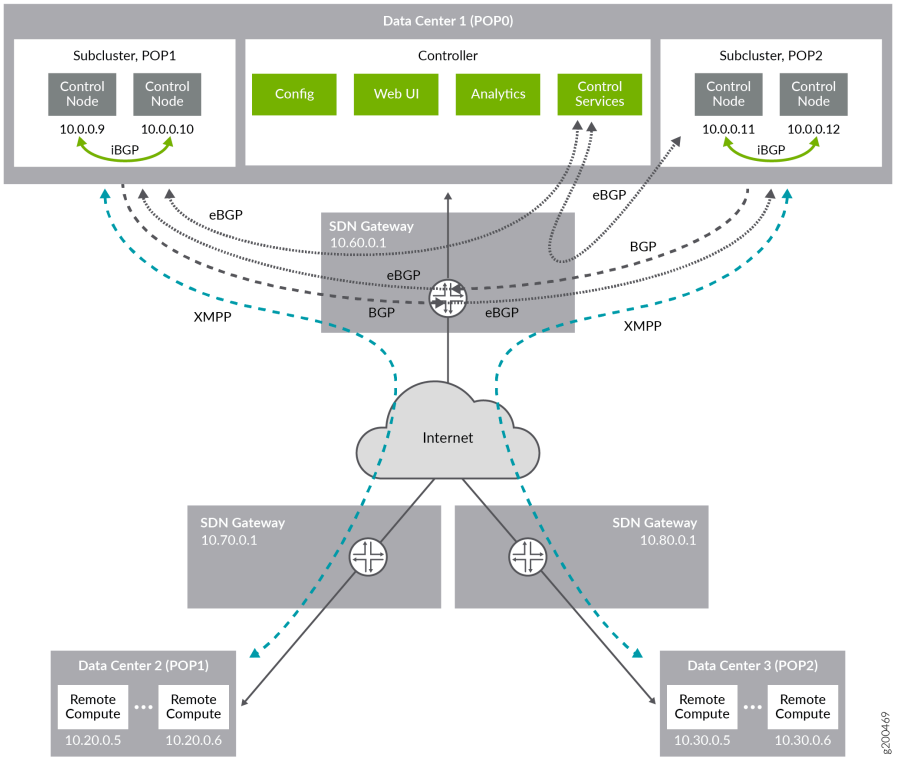
- SR-IOVによりVMはNICにダイレクトアクセス
- vRouterはBypassされる
- VMはSR-IOV, Kernel vRouterの併用化

SMART NIC VROUTER



- vRouter Forwarding機能はNICで提供される
- パフォーマンス向上
- ForwardingのためのCPUリソースを解放

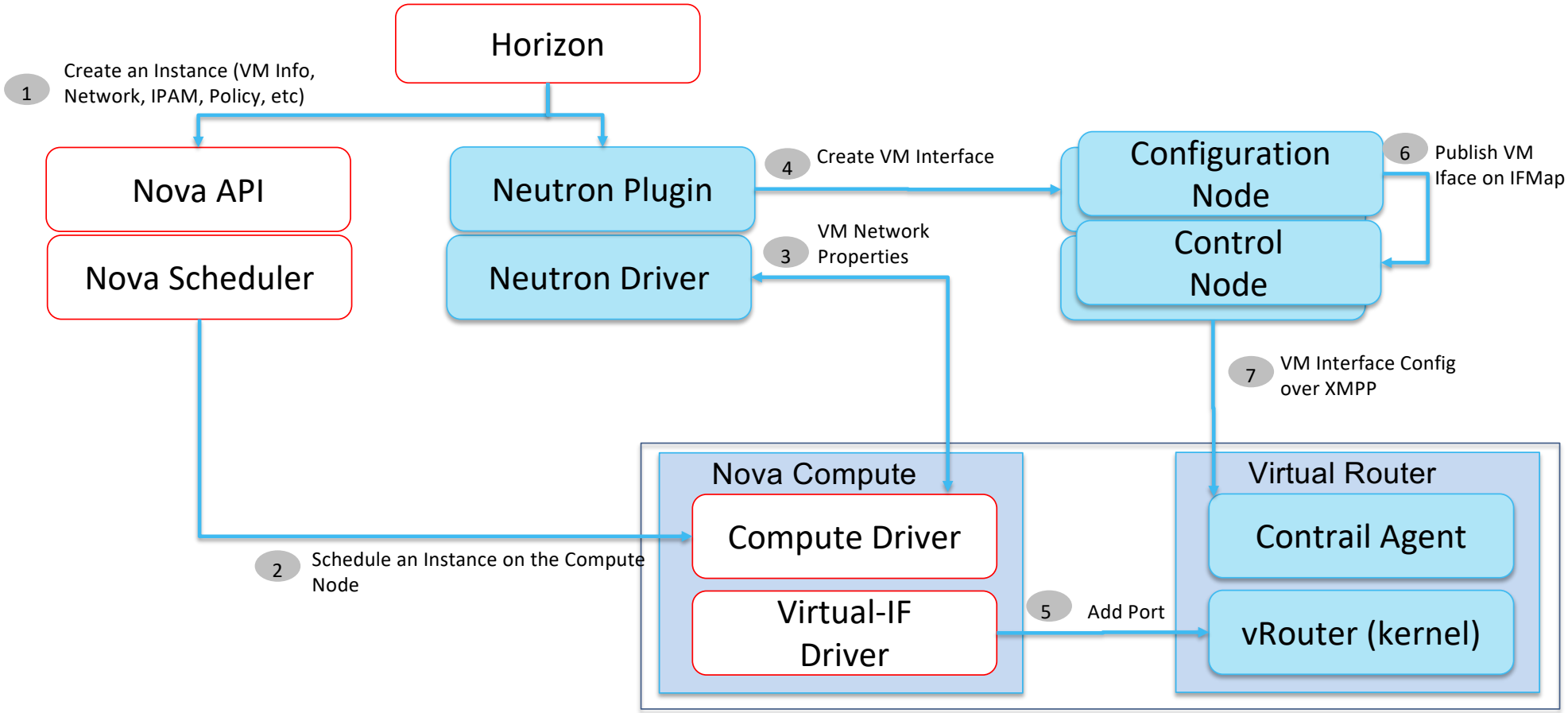
Contrail Remote Computeアーキテクチャ



- 複数のSmall DCへの拡張
- リモートサイトのComputeをSubClusterとして管理し、経路交換やコンフィグ情報を管理
- SubClusterはContrail ControllerのControl Nodeのみとなり、Config, WebUI, Analyticsは共有される
- SubCluster Control NodeはRemote ComputeとXMPP接続、Local GatewayとMP-eBGP接続を行う
- SubCluster Control Nodeは他のSubClusterとは直接BGP Peeringしない

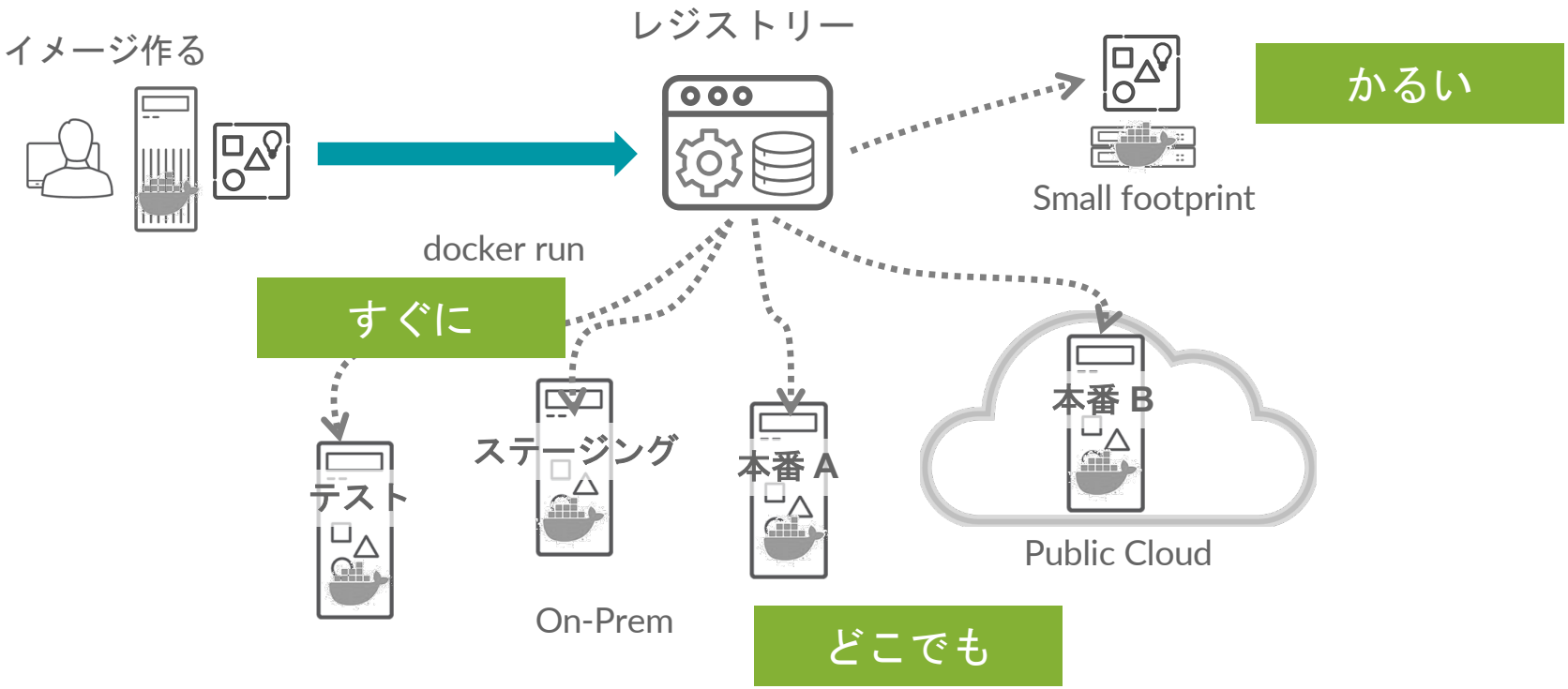
CEM + OpenStack

VM CREATION WORKFLOW



CEM + K8S

DOCKER/コンテナが使われる理由

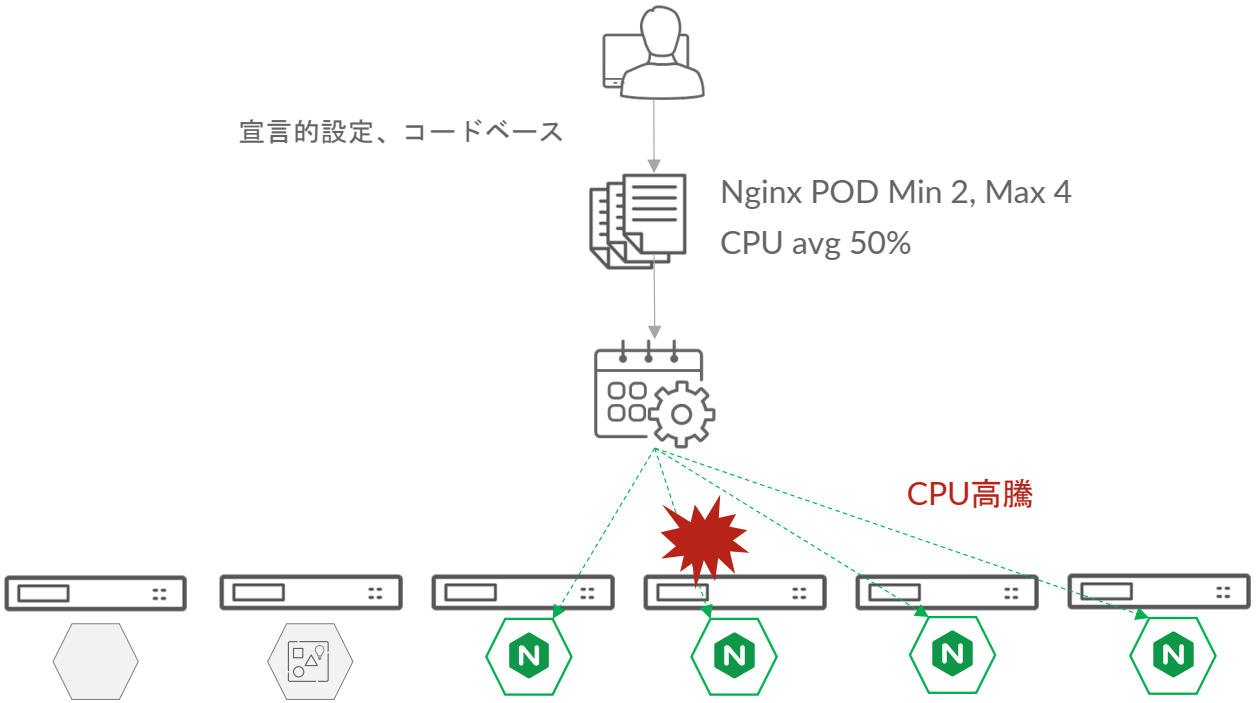


クラウドネイティブ

Kubernetes

- Infrastructure as Code (IaC)
- スケジューリング
- オートヒーリング
- オートスケーリング

etc...

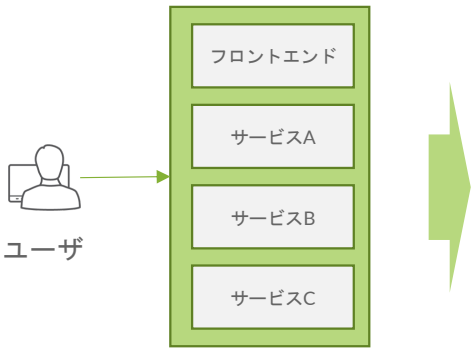


意図した構成を自動で維持する仕組みが備わっている

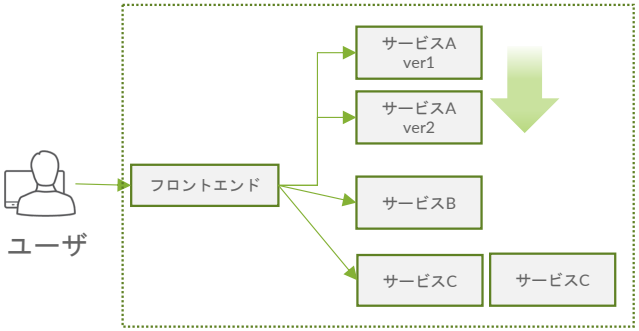
コンテナ化が進む１つの理由

デジタル変革により新しいサービスや機能を短期間で継続的にリリースしていく事が求められ、
多くの企業は、目まぐるしく変化する市場やニーズに柔軟に対応 していく必要が出てきている

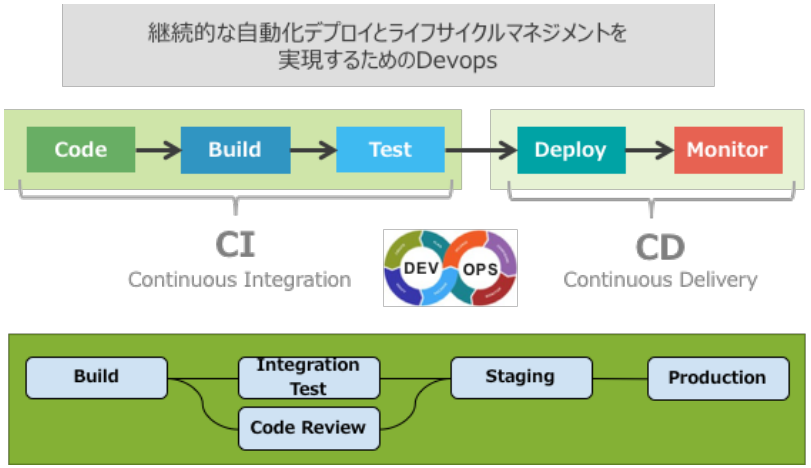
Monolithic application



Microservice application

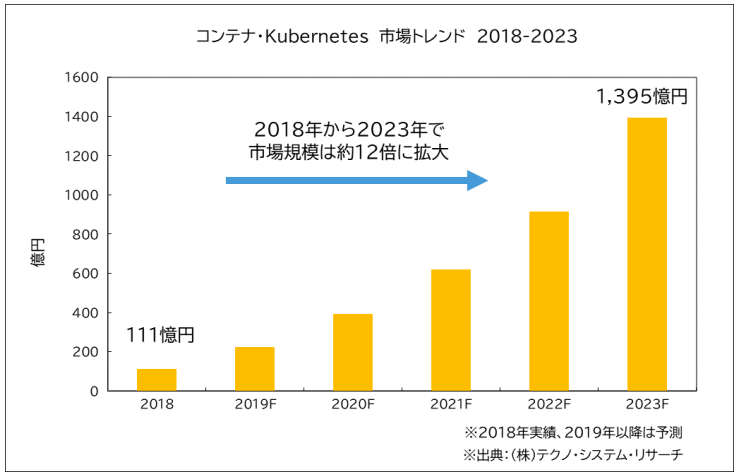


サービス毎に独立し疎結合
サービス毎のスケーリング
サービス単位の更新
サービス毎に異なる言語やデータベース



高頻度な更新：ビルド・テスト・デプロイ
起動・停止も高頻度
迅速なスケールアウト

国内コンテナ市場



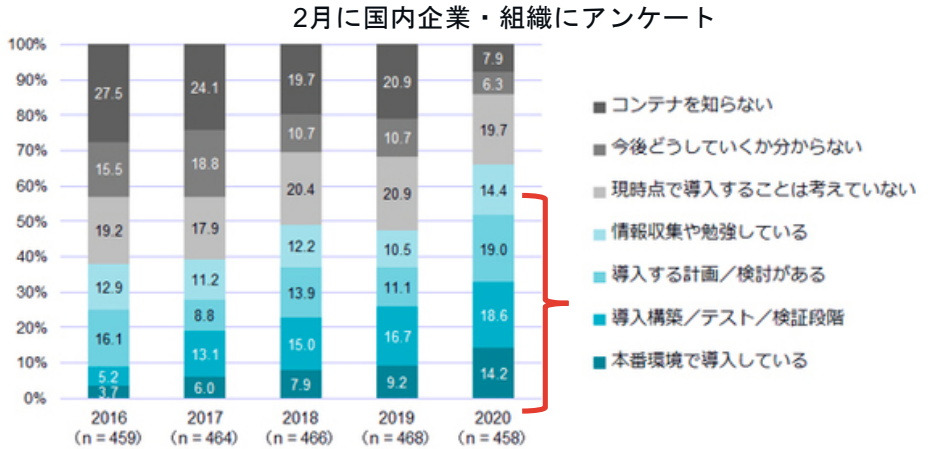
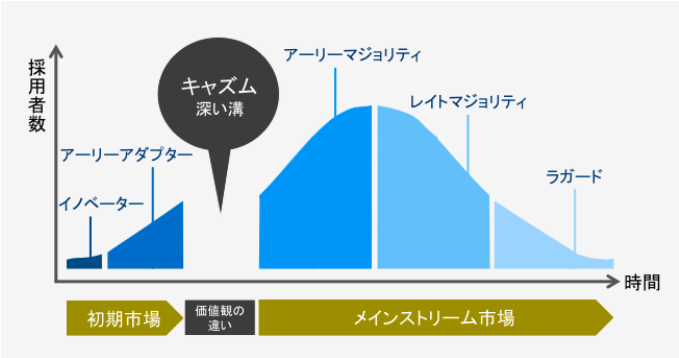


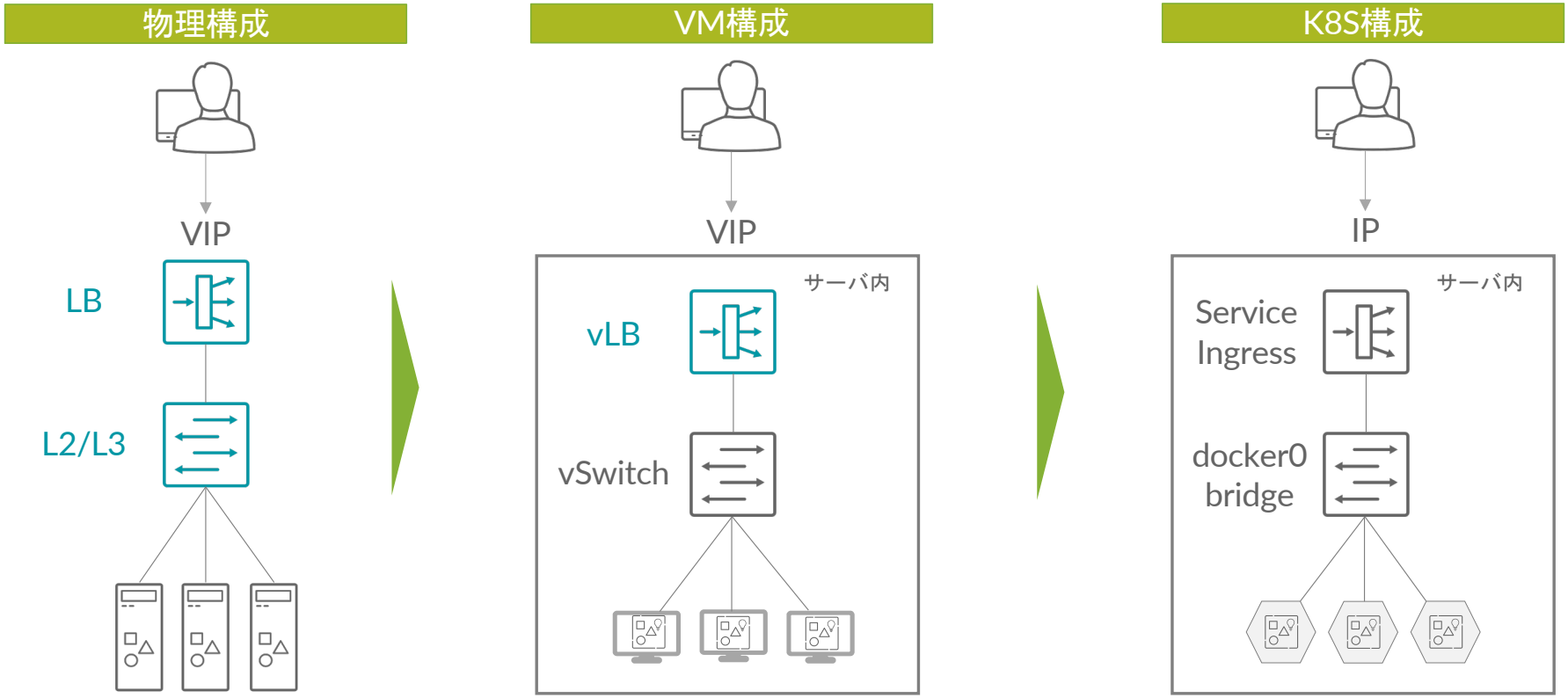
図1：コンテナの導入状況に関するユーザー調査結果（調査年別）（出典：IDC Japan）



「コンテナー導入はキャズム超えが目前。2020年内にはキャズムを超えることは確実であり、国内市場は本格的なコンテナーの普及期に突入していく」by IDC

今後多くの企業での導入が期待される

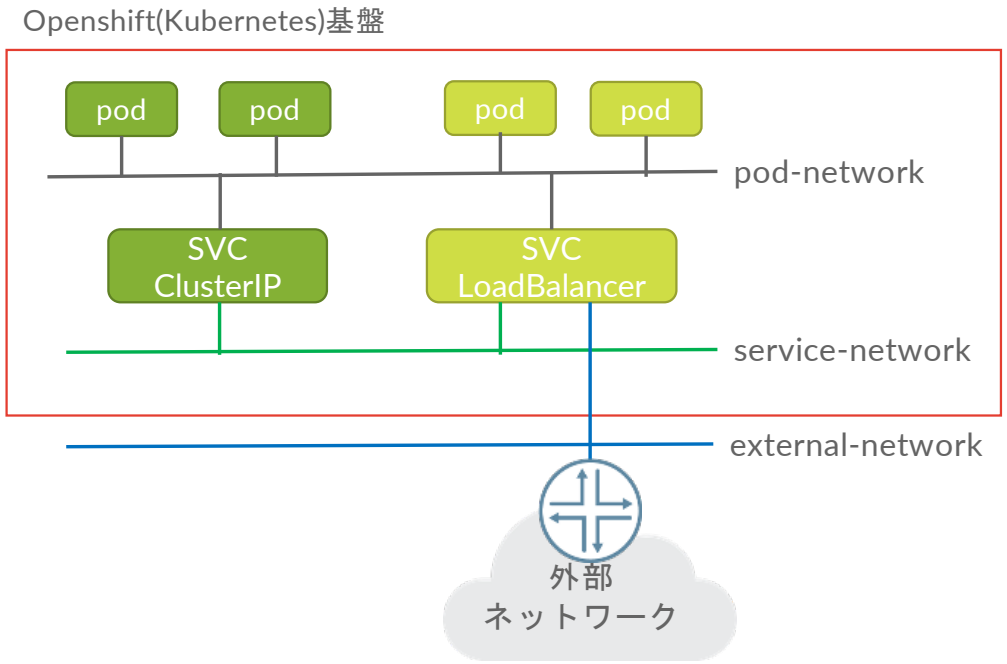
K8S ネットワーク



アプリケーションのスケールアウト・耐障害性を備えたネットワーク構成が基本

K8S ネットワーク

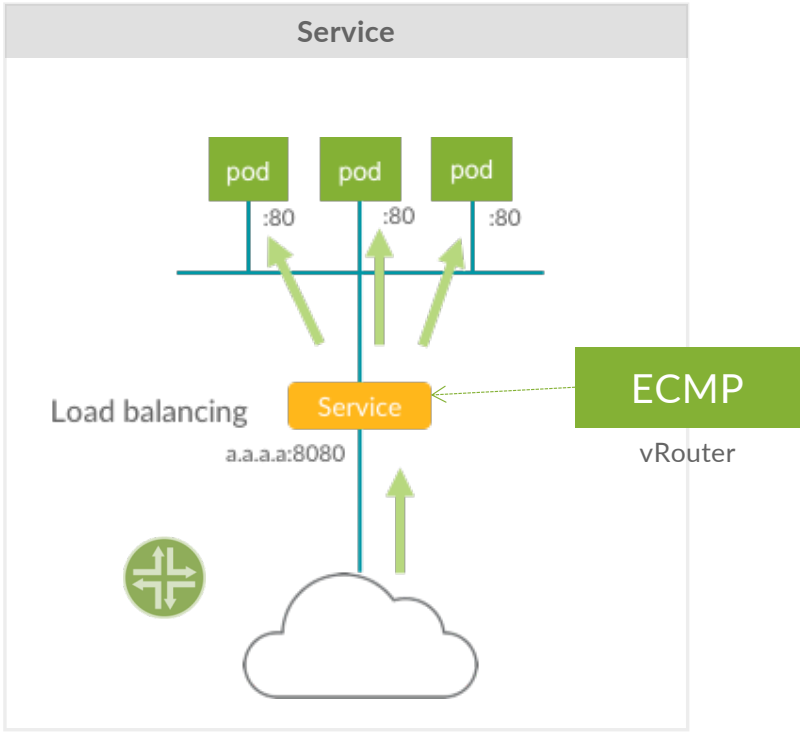
コンテナネットワークと種類

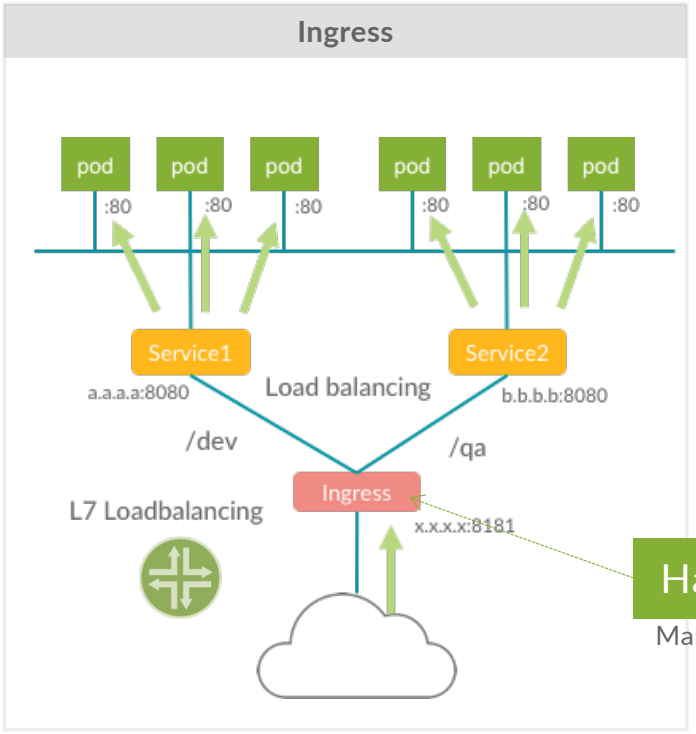


- pod-network
  POD (container)が接続する共通ネットワーク
- service network
  ”サービス”で利用するネットワーク
  PODはサービスに属し、サービスへの接続はこのネットワーク経由で行う
  Cluster IP
- external-network
  サービスに外部ネットワークから接続するのに利用する
  External IP

K8S + CONTRAILのネットワーク

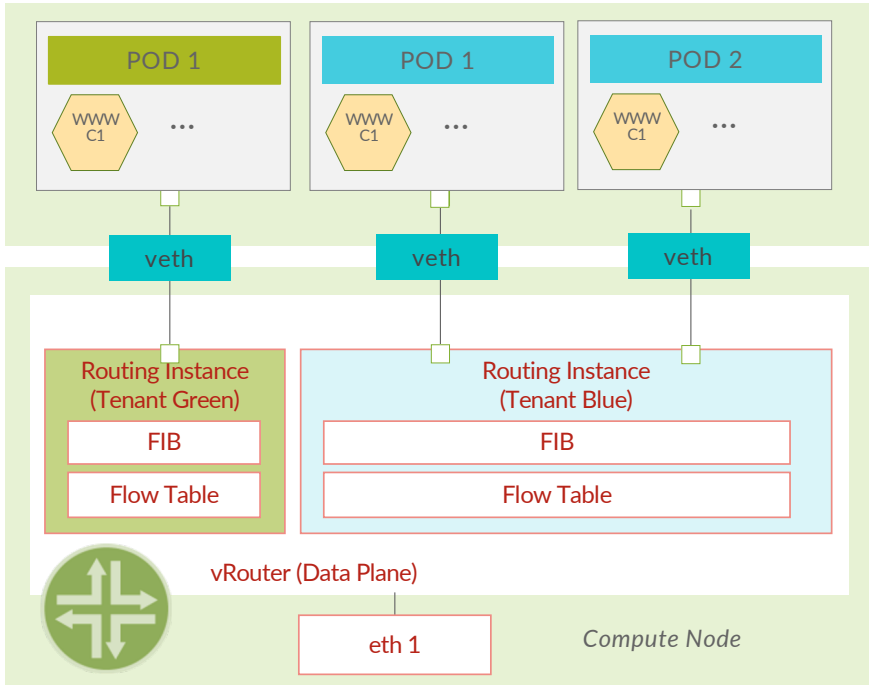
外部からの通信はOpenShit(Kubernetes)のようにロードバランシングの機能が提供される

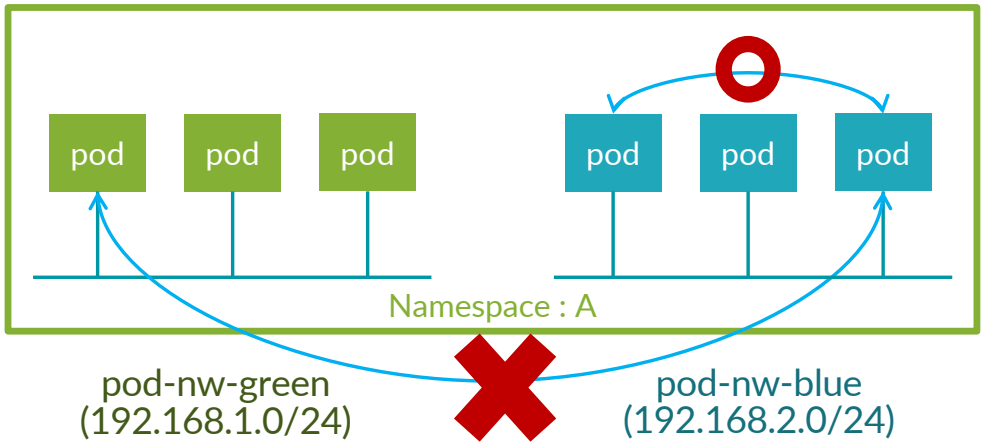




K8S + CONTRAILのネットワーク分離

仮想マシンネットワークのようにネットワークを分離
マルチテナント対応

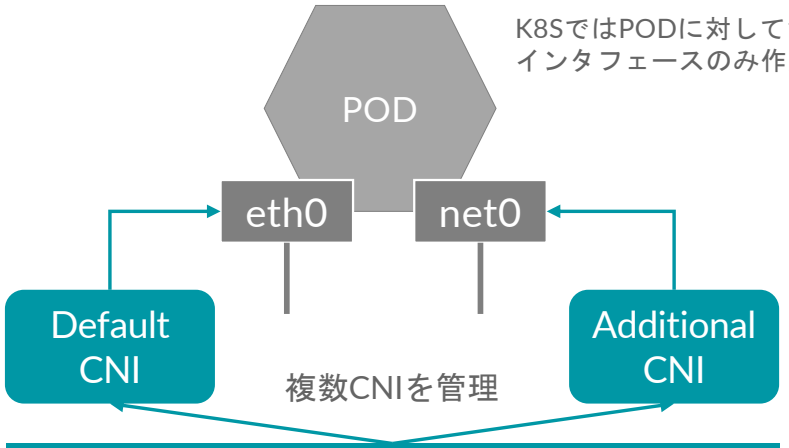




マルチインタフェース

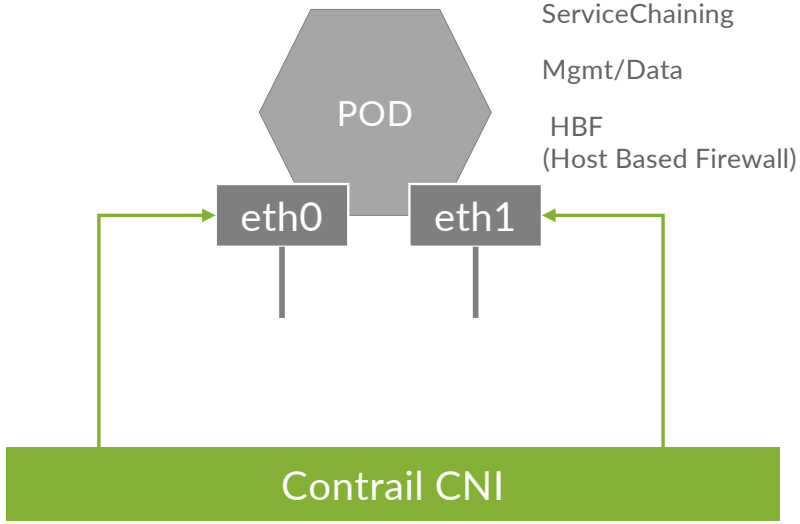
コンテナベースネットワークファンクションを考えるとマルチネットが必要 (e.g. CP/UP ネットワーク分離)

Multus CNI



Multus CNI は、他の CNI プラグインを呼び出すことのできる CNI プラグインです。これにより、他の CNI プラグインを使用して追加のネットワークインターフェースを作成できます。
(引用: REDHAT OpenShift)

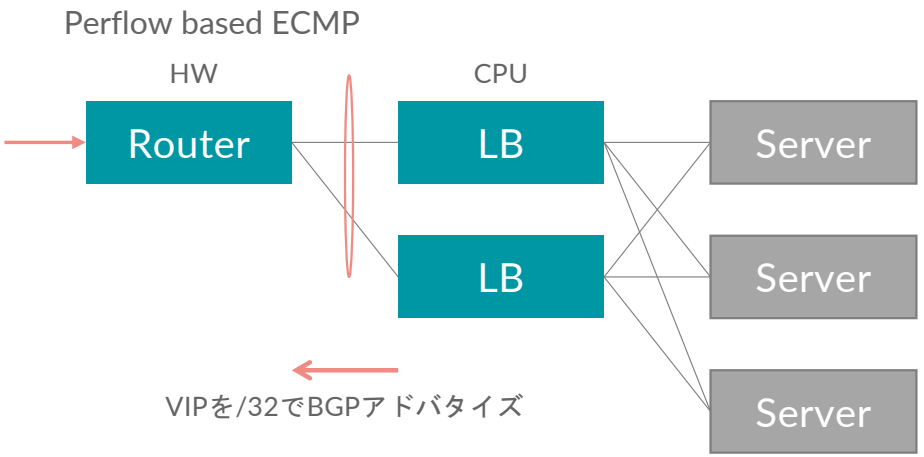
Contrail (multi-net)



Contrail のみで複数のネットワークインタフェースを作成

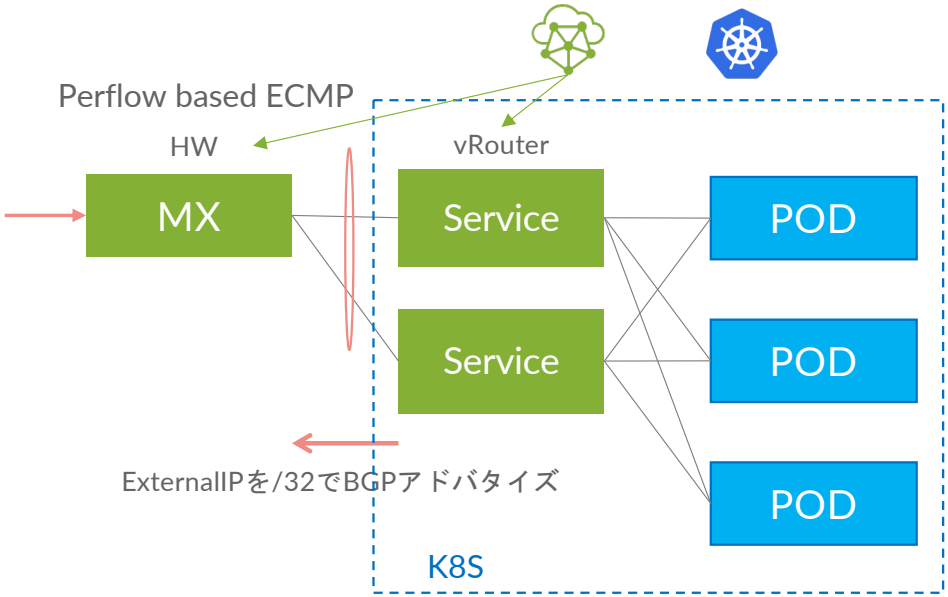
ハイパースケールDC LB アーキテクチャー

ハイパースケールDC LBアーキテクチャー



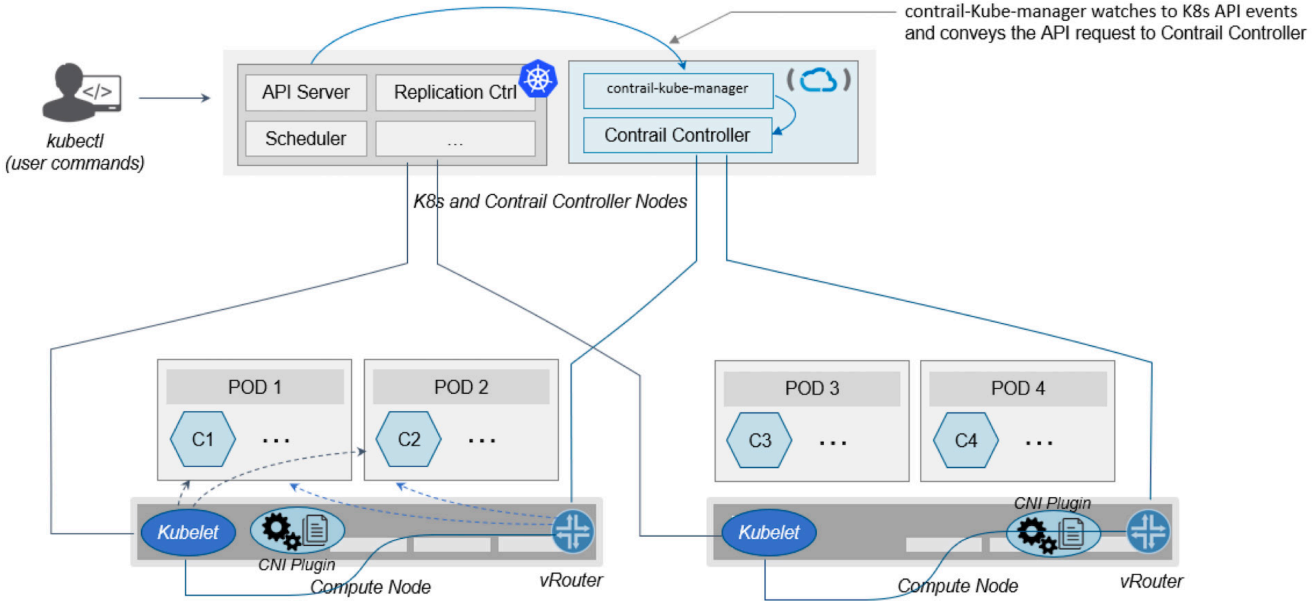
HWベースECMPを使ったスケールアウトモデル

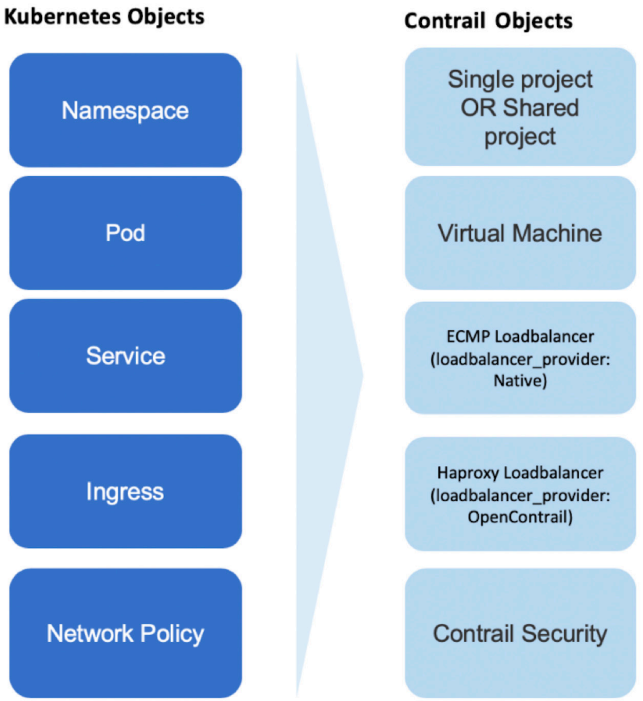
K8S x Contrai LBアーキテクチャー



HWベースECMPを使ったスケールアウトモデル

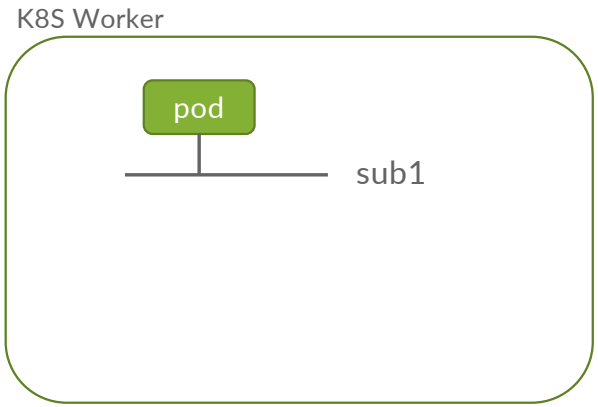
CEM + K8S アーキテクチャ & オブジェクトマッピング





K8S + CONTRAIL - ネットワーク作成(1)

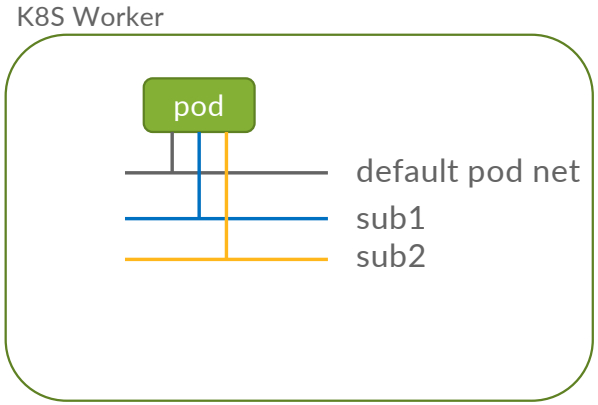
- Contrail側で作成したVirtual Networkを使用し、PODを作成
- AnnotationsにてContrail側のDomain, Project, Virtual Networkを指定
- 本手法は1POD 1 Virtual Networkとなり、Multi NICでは利用不可




```
apiVersion: v1
kind: Pod
metadata:
  name: cirros1
  annotations: {
     "opencontrail.org/network" : '{"domain":"default-
domain", "project": "k8s-default", "name": ”sub1"}’
  }
spec:
  containers:
  - name: cirros1
    image: docker.io/cirros
```

K8S + CONTRAIL - ネットワーク作成(2)

- Contrail側で作成したVirtual Networkを使用し、K8S上でNetworkAttachmentDefinitionを定義し、PODを作成
- AnnotationsにてContrail側のDomain, Project, Virtual Networkを指定
- Multi NIC PODを作成可能となり、Default pod-networkにも接続される



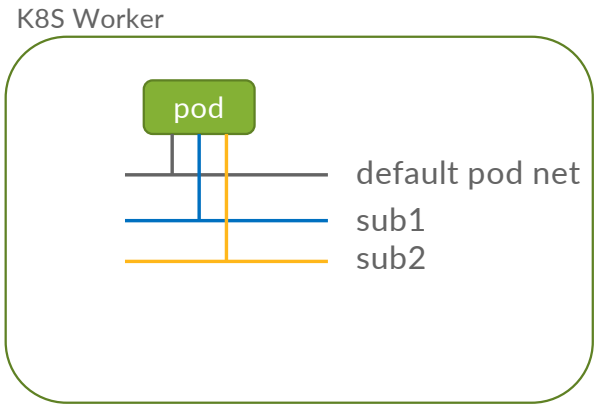

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: sub1
  annotations:
    "opencontrail.org/network" : '{"domain":"default-domain",
"project": "k8s-default", "name": "sub1"}'
spec:
  config: '{
    "cniVersion": "0.3.1",
    "type": "contrail-k8s-cni"
  }'
```

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: sub2
  annotations:
    "opencontrail.org/network" : '{"domain":"default-domain",
"project": "k8s-default", "name": "sub2"}'
spec:
  config: '{
    "cniVersion": "0.3.1",
    "type": "contrail-k8s-cni"
  }'
```

```
apiVersion: v1
kind: Pod
metadata:
  name: cirros1
  annotations: {
      k8s.v1.cni.cncf.io/networks: '[
        { "name": "sub1" },
        { "name": "sub2" }
      ]'
  }
spec:
  containers:
  - name: cirros1
    image: docker.io/cirros
```

K8S + CONTRAIL - ネットワーク作成(3)

- K8S上でNetworkAttachmentDefinitionを定義し、PODを作成
- AnnotationsにてContrail側のCIDR, Fabric Forwarding, SNATを指定
- Multi NIC PODを作成可能となり、Default pod-networkにも接続される



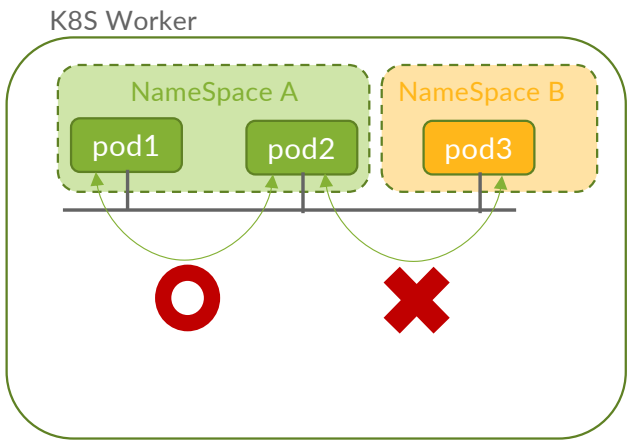

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
 annotations:
   opencontrail.org/cidr: "10.0.0.0/24"
   opencontrail.org/ip_fabric_forwarding: "false"
   opencontrail.org/ip_fabric_snat: "false"
 name: sub1
 namespace: default
spec:
 config: '{ "cniVersion": "0.3.0", "type": "contrail-k8s-cni" }'
```

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
 annotations:
   opencontrail.org/cidr: "20.0.0.0/24"
   opencontrail.org/ip_fabric_forwarding: "false"
   opencontrail.org/ip_fabric_snat: "false"
 name: sub2
 namespace: default
spec:
 config: '{ "cniVersion": "0.3.0", "type": "contrail-k8s-cni" }'
```

```
apiVersion: v1
kind: Pod
metadata:
 name: cirros1
 annotations: {
     k8s.v1.cni.cncf.io/networks: '[
       { "name": "sub1" },
       { "name": "sub2" }
    ]'
 }
spec:
 containers:
 - name: cirros1
   image: docker.io/cirros
```

K8S + CONTRAIL - NAMESPACE

- Namespace はPODのグループを定義し、異なるNamespace間の通信を制御する。
- K8S上で定義したNameSpaceはContrailのProjectとして管理される
- 同一Namespace間は通信できるが、異なるNamespace感は通信できない(isolation: true時)
- Podネットワークは共通で、PODは同一ネットワーク上に作られるがNamespaceで制御が可能




```
apiVersion: v1
kind: Namespace
metadata:
 name: ns_a
 annotations: {
  "opencontrail.org/isolation" : "true"
 }
```

```
apiVersion: v1
kind: Namespace
metadata:
 name: ns_b
 annotations: {
  "opencontrail.org/isolation" : "true"
 }
```

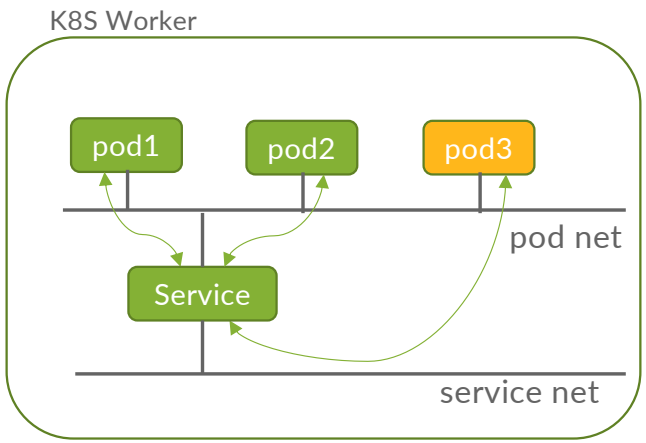
```
apiVersion: v1
kind: Pod
metadata:
  name: cirros1
  namespace: ns_a
spec:
  containers:
  - name: cirros1
    image: docker.io/cirros
```

```
apiVersion: v1
kind: Pod
metadata:
  name: cirros2
  namespace: ns_a
spec:
  containers:
  - name: cirros1
    image: docker.io/cirros
```

```
apiVersion: v1
kind: Pod
metadata:
  name: cirros3
  namespace: ns_b
spec:
  containers:
  - name: cirros1
    image: docker.io/cirros
```

K8S + CONTRAIL – SERVICE(CLUSTER IP)

- Service(ClusterIP)を定義することにより、ECMPベースで複数PODにアクセスが可能
- Contrail環境ではpod networkとservice networkは互いにRouteLeakしているため、PODからClusterIPへ接続可




```
apiVersion: v1
kind: Service
metadata:
 name: web
spec:
 selector:
  app: web
 ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

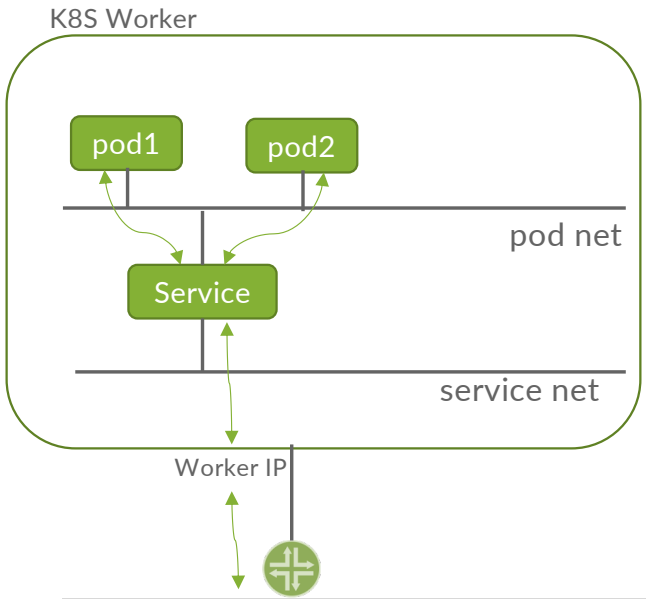
```
apiVersion: v1
kind: Pod
metadata:
 name: nginx1
 labels:
  app: web
spec:
 containers:
 - name: nginx1
   image: nginx:1.7.5
   ports:
    - containerPort: 80
```

```
apiVersion: v1
kind: Pod
metadata:
 name: nginx2
 labels:
  app: web
spec:
 containers:
 - name: nginx2
   image: nginx:1.7.5
   ports:
    - containerPort: 80
```

```
# kubectl get endpoints
NAME      ENDPOINTS          AGE
web       10.47.255.249:80   9m45s

# kubectl get services
NAME      TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
web       ClusterIP   10.100.139.201  <none>        80/TCP    9m54s
```

K8S + CONTRAIL – SERVICE(NODE PORT)

- Service(NodePort)を定義することにより、WorkerのIPでPortForwardingが可能




```
apiVersion: v1
kind: Service
metadata:
 name: nodeport1
spec:
 selector:
  app: web
 type: NodePort
 ports:
 - targetPort: 80
   port: 80
   nodePort: 32001
```

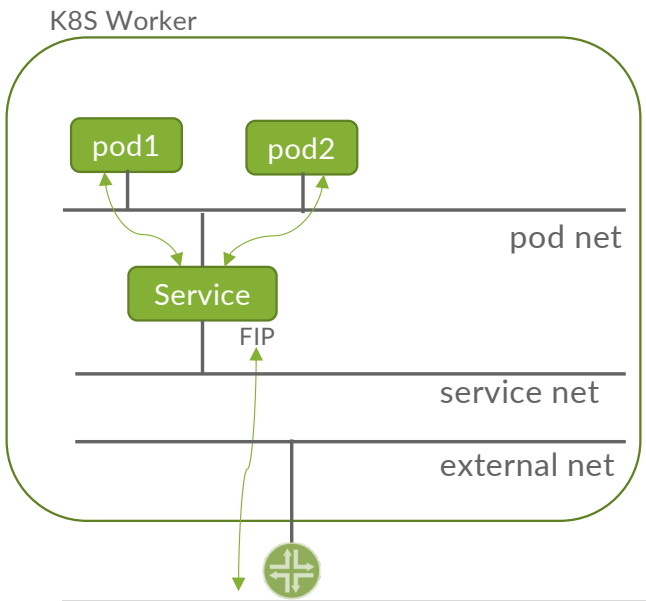
```
apiVersion: v1
kind: Pod
metadata:
 name: nginx1
 labels:
   app: web
spec:
 containers:
 - name: nginx1
   image: nginx:1.7.5
   ports:
     - containerPort: 80
```

```
apiVersion: v1
kind: Pod
metadata:
 name: nginx2
 labels:
   app: web
spec:
 containers:
 - name: nginx2
   image: nginx:1.7.5
   ports:
     - containerPort: 80
```

```
kubectl get endpoints
NAME        ENDPOINTS          AGE
nodeport1   10.47.255.249:80   15m
kubectl get services
NAME        TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)       AGE
nodeport1   NodePort    10.101.149.64   <none>        80:32001/TCP  17m
```

K8S + CONTRAIL – SERVICE(LOAD BALANCER)

- Service(LoadBalancer)を定義することにより、FloatinIPを使用し外部からPODへECMPアクセス可能




```
apiVersion: v1
kind: Service
metadata:
  name: web
spec:
  selector:
    app: web
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: LoadBalancer
```

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx1
  labels:
    app: web
spec:
  containers:
  - name: nginx1
    image: nginx:1.7.5
    ports:
      - containerPort: 80
```

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx2
  labels:
    app: web
spec:
  containers:
  - name: nginx2
    image: nginx:1.7.5
    ports:
      - containerPort: 80
```

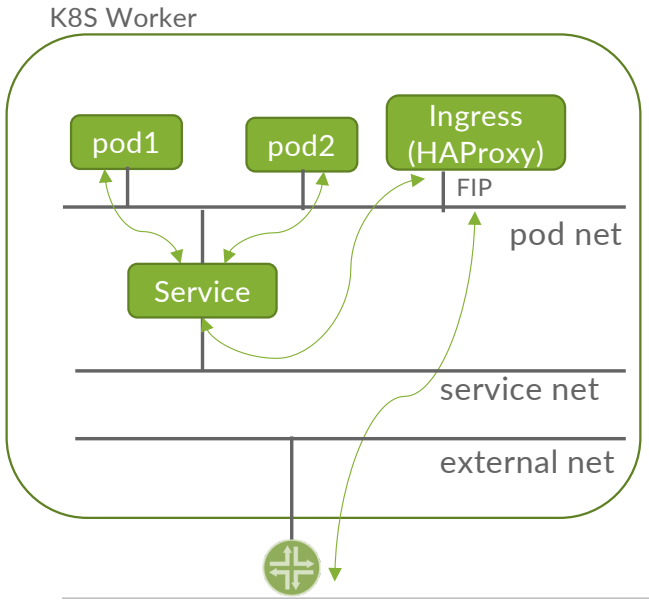
```
# kubectl get endpoints
NAME      ENDPOINTS          AGE
web       10.47.255.249:80   42s

# kubectl get services
NAME      TYPE          CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
web       LoadBalancer  10.109.75.19    88.88.88.3    80:32737/TCP   36s
```

K8S + CONTRAIL – INGRESS

- Ingressを定義することによりL7 LoadBalancingが可能
- Contrail環境ではHAProxyがActive/Standbyでデプロイされ、Floating IPが付与される

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress1
spec:
  backend:
    serviceName: web
    servicePort: 8080
```

```
apiVersion: v1
kind: Service
metadata:
  name: web
spec:
  ports:
  - port: 8080
    targetPort: 80
  selector:
    app: web
```

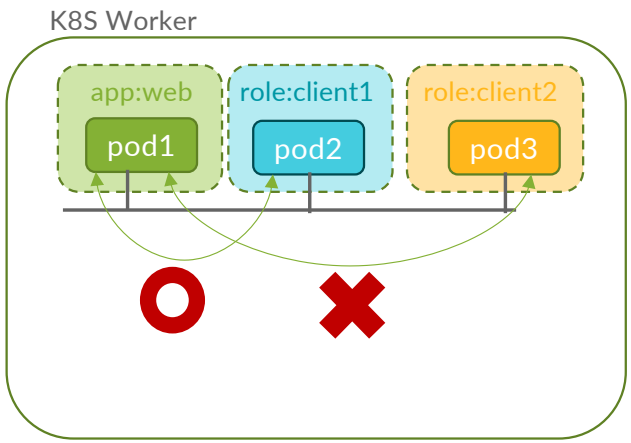



```
# kubectl get endpoints
NAME      ENDPOINTS          AGE
web       10.47.255.246:80   3m56s

# kubectl get services
NAME      TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
web       ClusterIP   10.98.197.94   <none>        8080/TCP   4m18s

# kubectl get ingress
NAME      HOSTS   ADDRESS                  PORTS   AGE
ingress1  *       10.47.255.249,88.88.88.3 80      7m33s
```

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx1
  labels:
    app: web
spec:
  containers:
  - name: nginx1
    image: nginx:1.7.5
    ports:
      - containerPort: 80
```

K8S + CONTRAIL – NETWORK POLICY

- Network Policyを定義することにより、Ingress/EgressのFilteringが可能
- Namespace, Label, IP Address, TCP/UDP PortベースのFilteringが可能
- Contrail環境では動的にFirewall Policyが設定される
- 以下はLableベースのFilteringの例



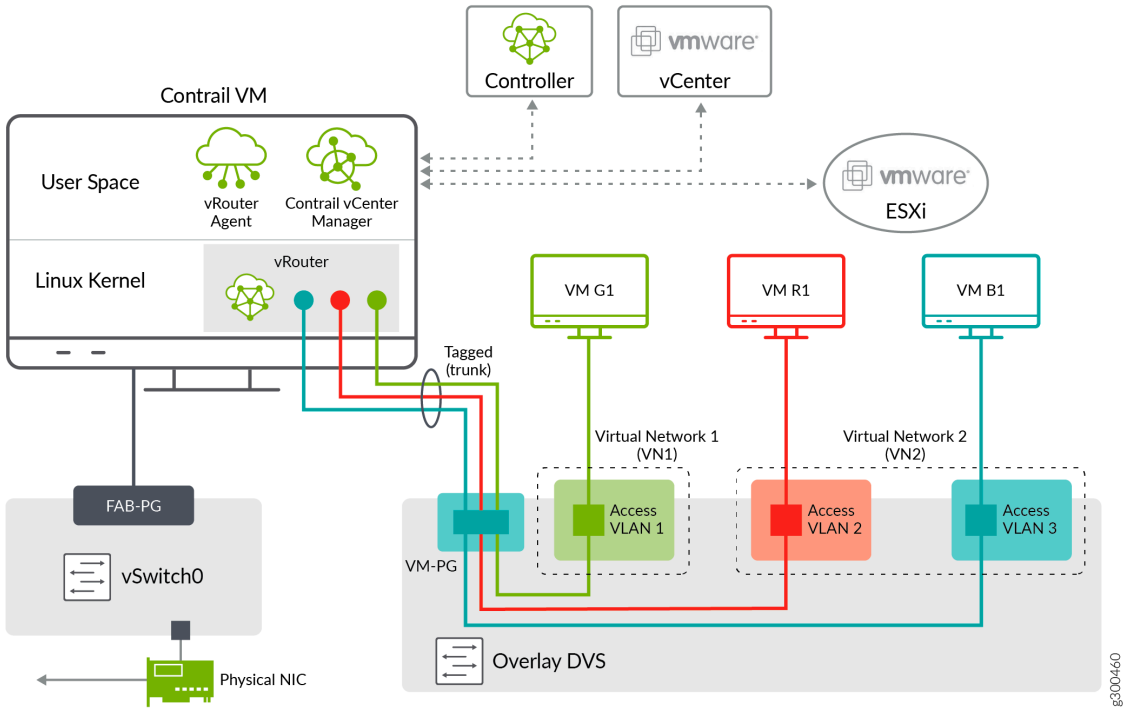

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: policy1
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: web
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: client1
```

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx1
  labels:
    app: web
spec:
  containers:
  - name: nginx1
    image: nginx:1.7.5
    ports:
      - containerPort: 80
```

```
apiVersion: v1
kind: Pod
metadata:
  name: client1
  labels:
    role: client1
spec:
  containers:
  - name: clinet1
    image: centos
    command: [ "/bin/bash", "-c" ]
    args: [ "sleep infinity" ]
    securityContext:
      privileged: true
```

CEM + vCenter

VROUTER ON ESXI – SVS/DVS FOR UPLINK



インプリ:

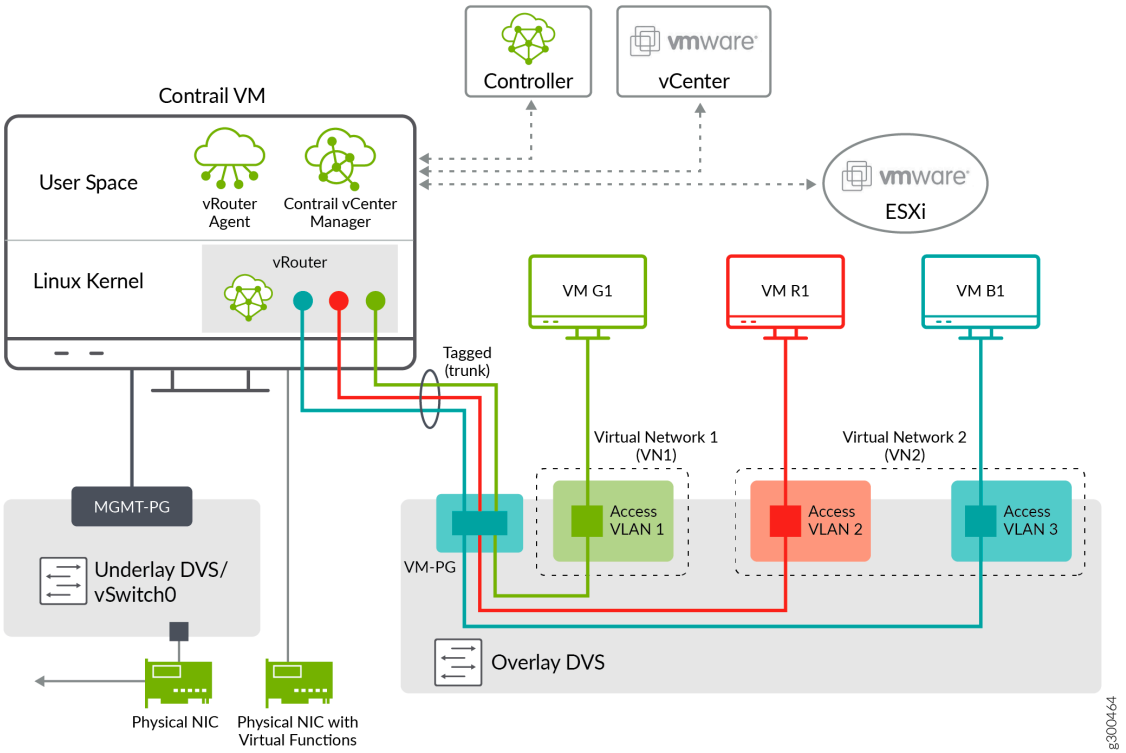
- Contrail VM(vRouter)はVMとして各ESXi上で稼働
- vRouter Uplink(VXLAN Tunnel, Control)はSVS, DVSどちらでも可能
- vRouter Downlink(Overlay側)はDVS必須
- VM毎にVLANが付与され、vRouterはDVSとTrunk接続されている
- VM間通信はvRouterを必ず経由するためMicroSegmentationが可能

機能:

- 既存のvCenter環境におけるインストレーションとプロビジョニングが可能
- IPAM (IPアドレス管理)
  - DHCP
  - DNS
- Network Orchestration(Virtual Networks):
  - 複数のVNを作成してVMに紐付ける

- 異なるVN間でのVM向けポリシー接続
- サービス・チェイニング
- Security Groups
- フローティング IP

VROUTER ON ESXI – SR-IOV FOR UPLINK



インプリ:

- Contrail VM(vRouter)はVMとして各ESXi上で稼働
- vRouter Uplink(VXLAN Tunnel, Control)はSR-IOVを使用
- vRouter Downlink(Overlay側)はDVS必須
- VM毎にVLANが付与され、vRouterはDVSとTrunk接続されている
- VM間通信はvRouterを必ず経由するためMicroSegmentationが可能

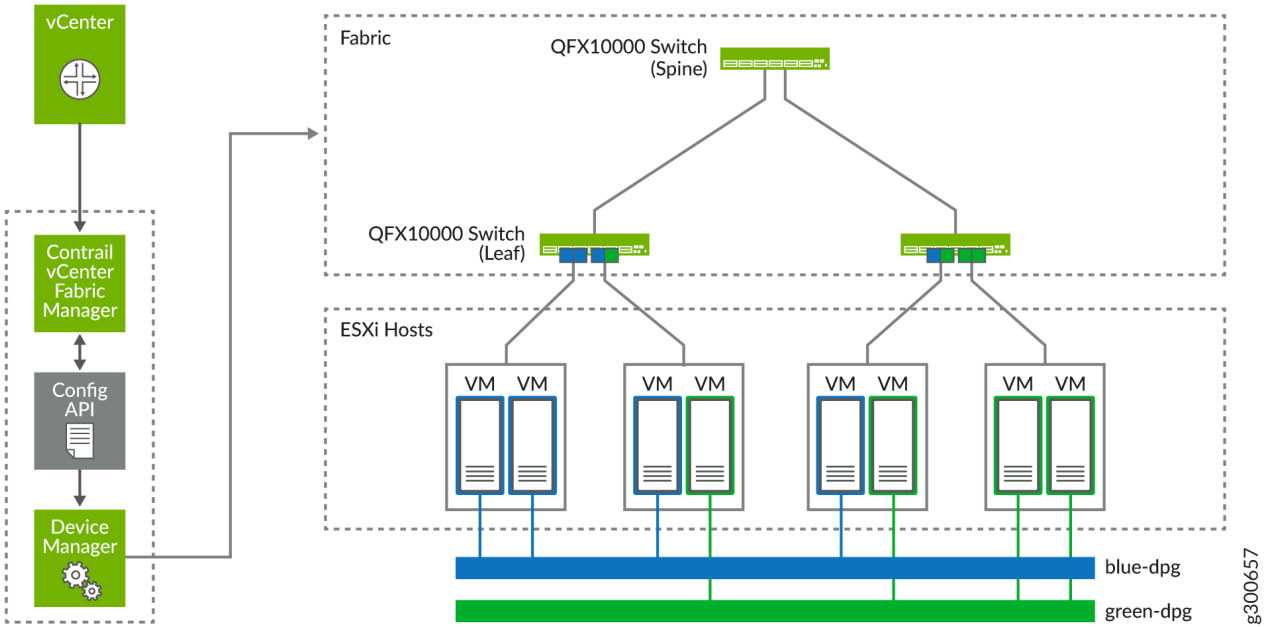
機能:

- 既存のvCenter環境におけるインストレーションとプロビジョニングが可能
- IPAM (IPアドレス管理)
  - DHCP
  - DNS
- Network Orchestration(Virtual Networks):
  - 複数のVNを作成してVMに紐付ける

- 異なるVN間でのVM向けポリシー接続
- サービス・チェイニング
- Security Groups
- フローティング IP

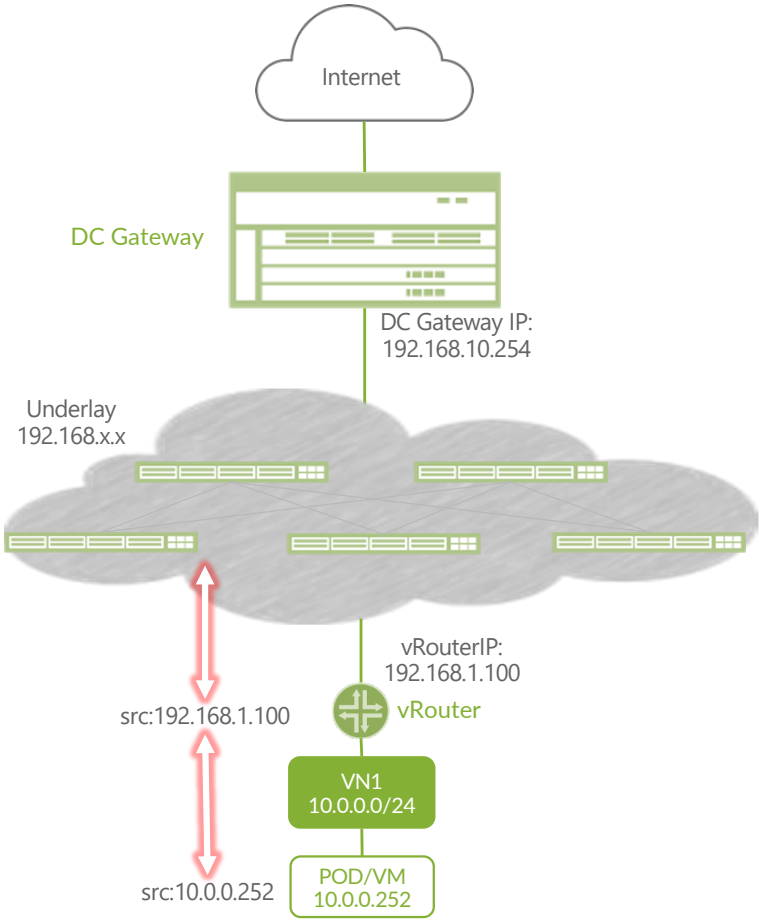
CONTRAIL VCENTER FABRIC MANAGER (CVFM)

- CVFMはvRouterを使用せずにvCenterと連携し、 VMware Distributed Port Group(DPG)の設定をToR(Leaf) SWに同期することが可能
- FabricでのVM通信の可視化、Path管理(Roadmap)が可能
- NSXがない、Leafに全VLAN Trunkできない環境下で有効

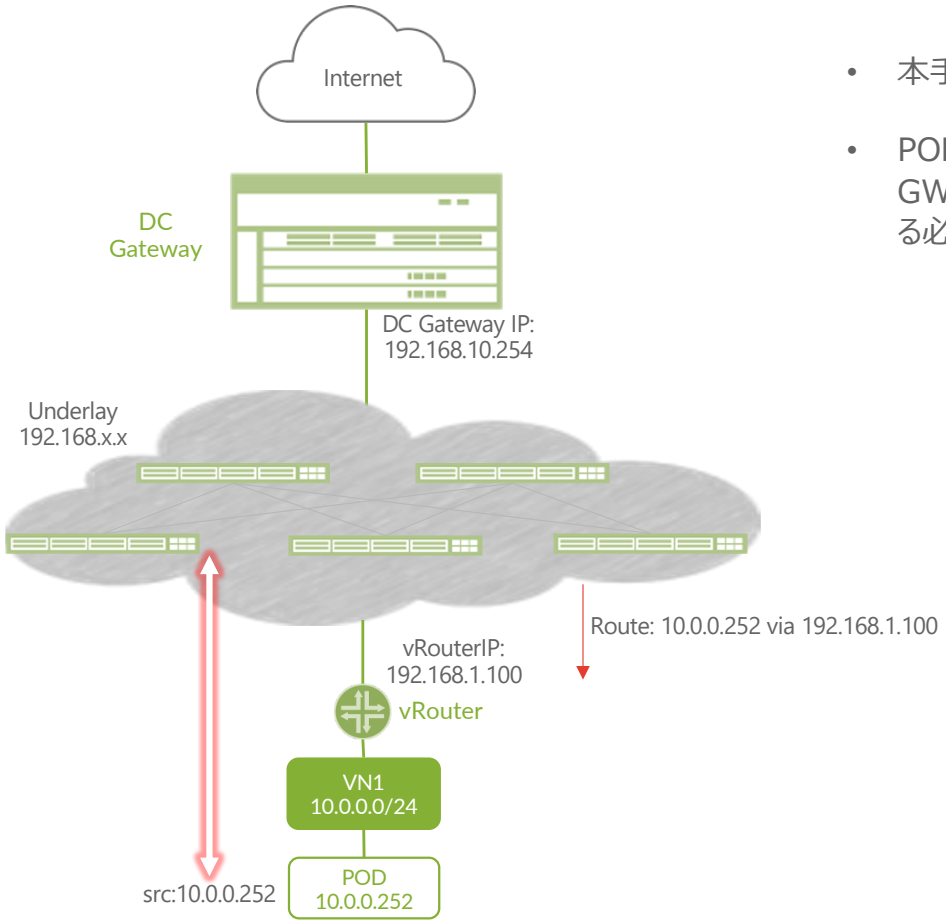


CEM詳細

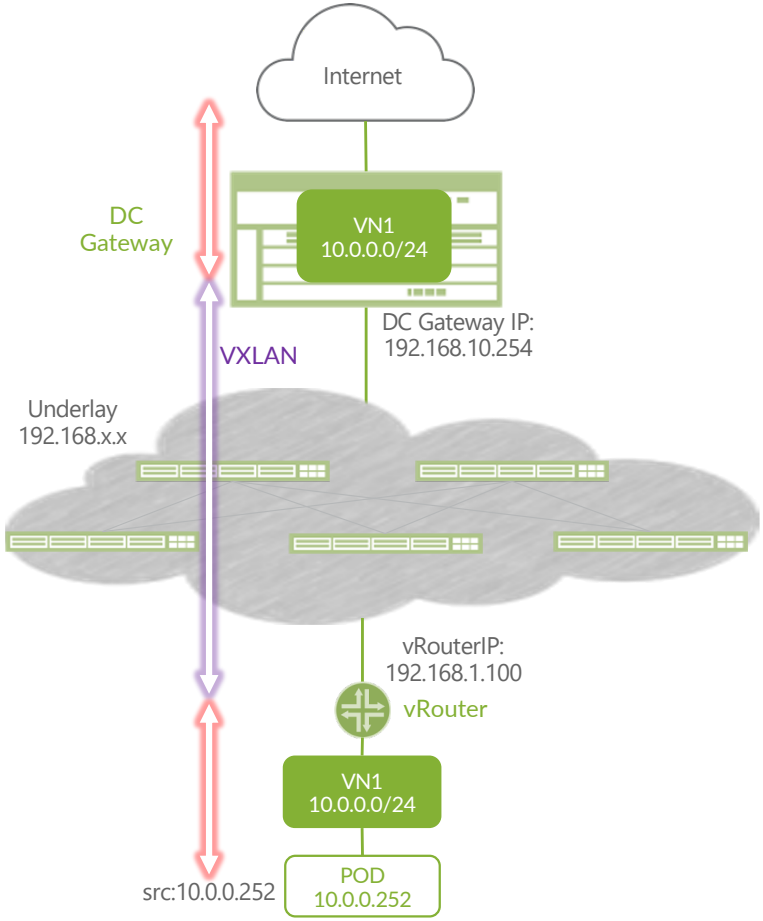
Contrail Networking



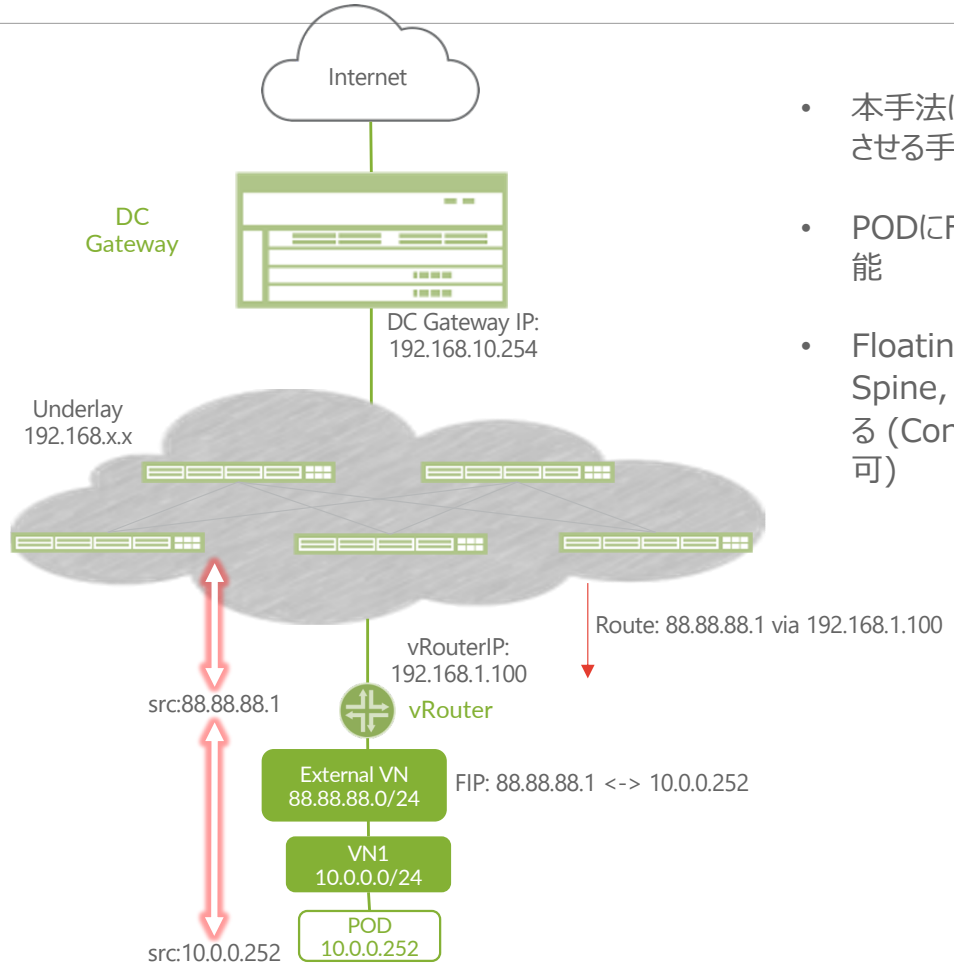
- 本手法はPOD Network(Overlay Net)から外部へ接続する際、vRouterでSNATしUnderlayへ接続する手法
- SNATのため外部からPOD/VMへのアクセスは不可
- vRouter IPでNATされるため外部からvRouter IPへ疎通が取れる必要がある



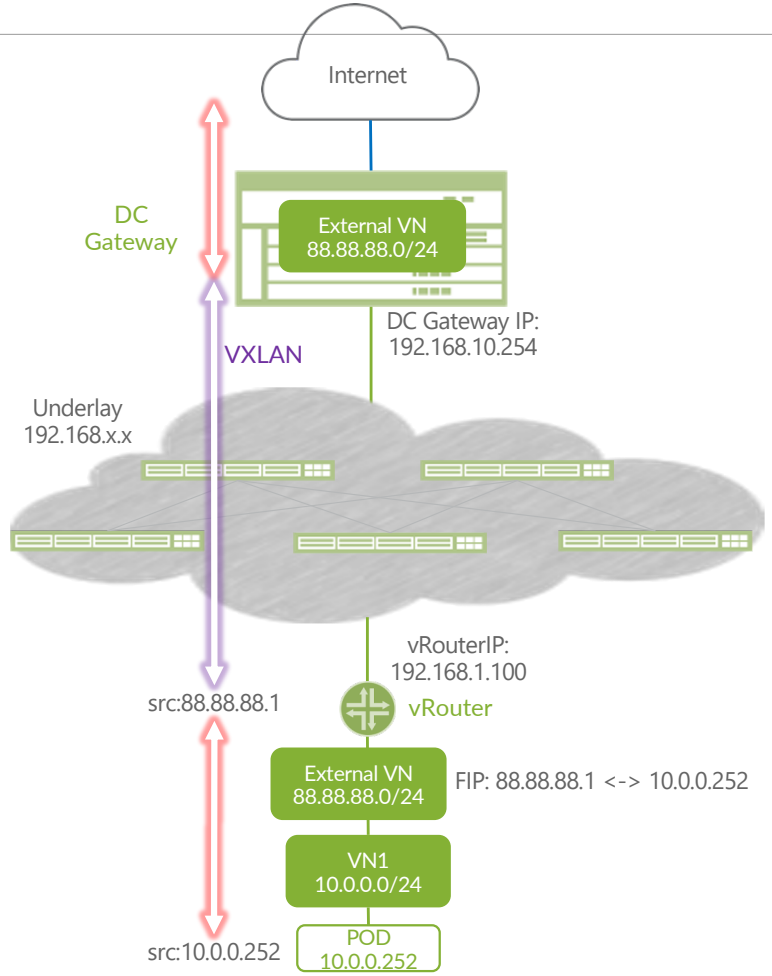
- 本手法はVirtual NetworkをvRouterから直接Underlayへ接続させる手法
- POD/VMで使用しているIPが直接Underlayへ接続されるため、外部RT, DC GW, Spine, LeafではPOD NetworkへのNextHopをvRouter IPに指定する必要がある



- 本手法はVirtual NetworkをDC GWへBGPで広報し、vRouter/DC GW間でVXLAN接続し、DC GWでUnderlayへ接続させる手法
- vRouter/DC GW間はVXLAN接続されるため、Spine/LeafではVirtual Networkのroute情報は不要



- 本手法はFloating IP Network(Virtual Network)を直接Underlayへ接続させる手法
- PODにFloating IPを付与し外部接続を行うため、外部からPODへもアクセス可能
- Floating IPが直接Underlayへ接続されるため、外部RT, DC GW, Spine, LeafではFloatingIPへのNextHopをvRouter IPに指定する必要がある (Contrail ControllerからUnderlay RouterへFIP RouteのAdvertiseも可)



- 本手法はFloating IP Network(Virtual Network)をDC GWへBGPで広報し、vRouter/DC GW間でVXLAN接続し、DC GWでUnderlayへ接続させる手法
- vRouter/DC GW間はVXLAN接続されるため、Spine/Leafでは FloatingIP Networkのroute情報は不要

CEM詳細

Fabric Automation

イーサネットファブリック vs IP Fabric + EVPN/VXLAN

Ethernet Fabric

シンプルだがベンダーロックされたブラックボックス

ベンダーロックイン
Ethernet Fabricは単一ベンダーでの構成が必須

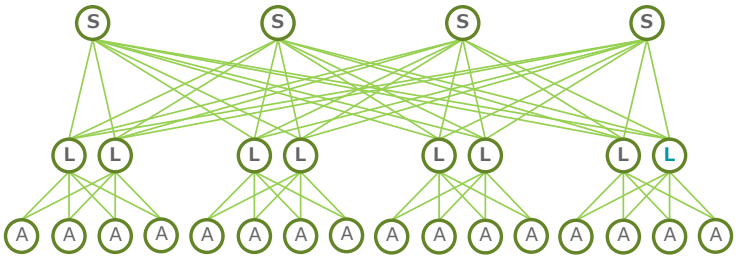
スケーラビリティ
1つのFabricを構成できるスイッチの台数が多くない

トラフィック制御
Ethernet Fabricはブラックボックスでトラフィックの制御が難しい

障害影響
ファブリック動作の場合、障害やメンテナンスが全体に影響してしまう

アップグレード問題
Ethernet Fabricは同一バージョンで構成が必須
個別アップグレードが難しい

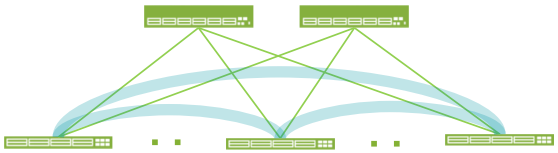
IP Clos Fabric



Open/Standard　スケーラブル　トラフィック制御

＋

EVPN / VXLAN



Open/Standard　スケーラブル　オーバーレイ

IPファブリック/EVPN/VXLAN採用の利点

IPファブリックの利点

従来のソリューション	IP Fabric
■ベンダーロックイン ベンダー固有のソリューション	■オープンスタンダード 標準ルーティングプロトコルをベースにしている
■スケーラビリティ Fabric内のスイッチ数が限られる(十数台程度が多い)	■ルーティングプロトコル接続のためスケーラブルで拡張しやすい
■トラフィック制御 Fabric内部の通信制御はできない	■ルーティング(メトリックなど)での通信制御が可能
■一体型アーキテクチャ 障害影響が全体に影響しやすい	■分散型のアーキテクチャ 障害は全体へ影響しにくい
■バージョンアップ 同一バージョンでの構成が必要 アップグレードがFabric全体に影響	■ルーティングプロトコル接続のためバージョンに依存しない アップグレードは各デバイス毎に実施

EVPN/VXLAの利点

何故VXLANなのか？

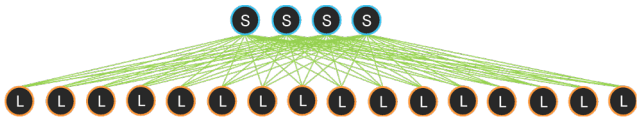
- レガシーシステムはL2の接続性を要求するため、IP Fabric上にVXLANを構築し、延伸を実現します。
- 標準プロトコルで多くのベンダーが採用しておりベンダー依存しません。

何故EVPNなのか？

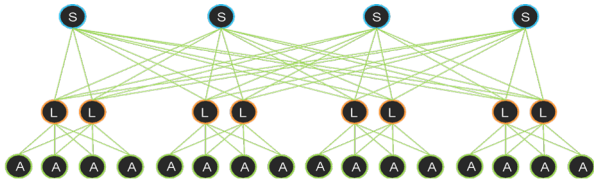
- マルチポイントでのL2トンネルを構築しARPの抑制やActive/Activeマルチホームなど有用な多くの機能を備えています。
- BGPベースであり、スケール（拡張性）があります。
- DCを跨いだ拡張が可能です。
- マルチテナントに対応可能です。
- 標準プロトコルで多くのベンダーが採用しておりベンダー依存しません。

IP FABRIC + EVPN/VXLANの課題

3-Stage Clos
Spine and Leaf



EVPN-VXLAN Fabric
PODs



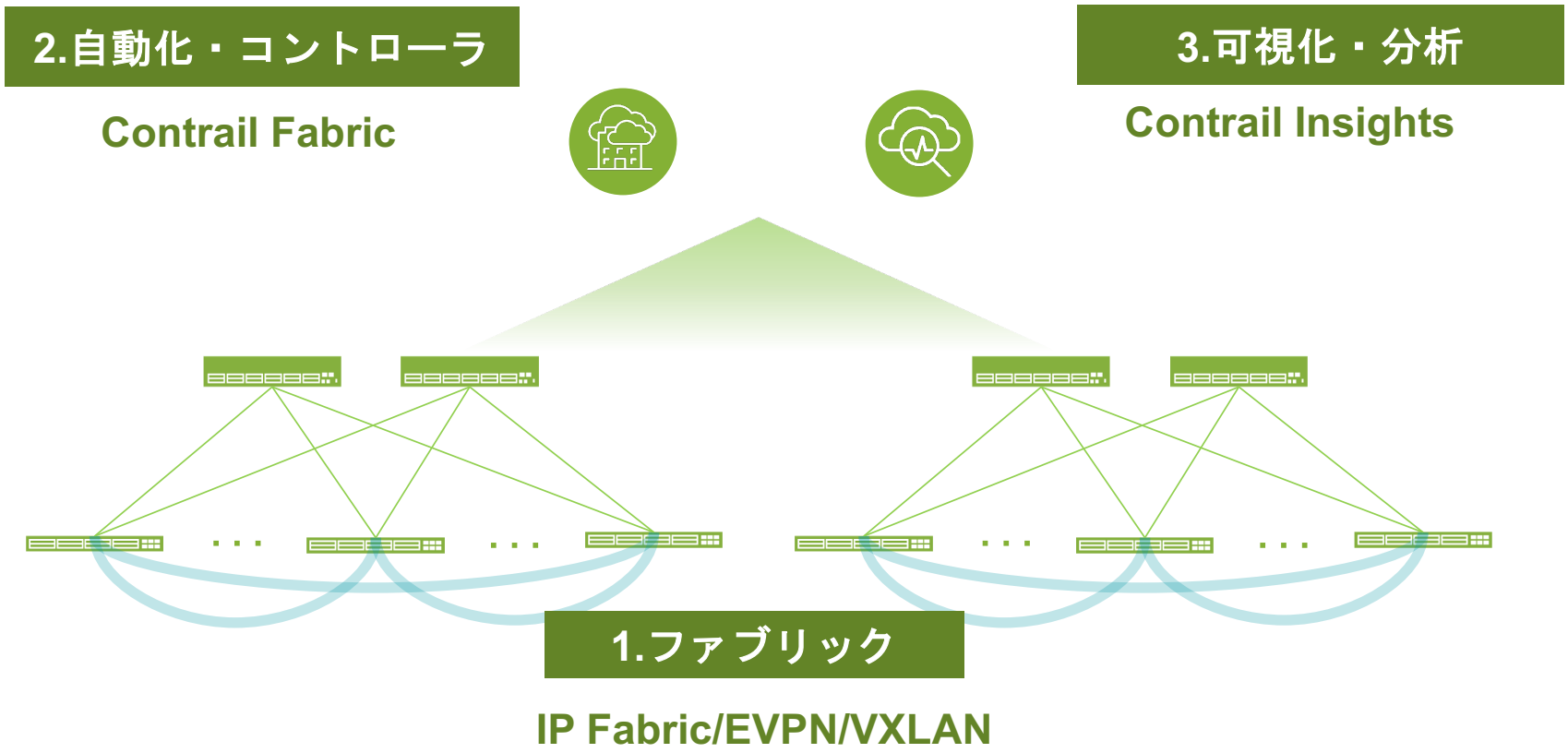
Interface Assignments
- IP addressing
- Loopback addressing
- Subnet masks
- PTP Links
- Server VLAN
- RVI assignment

Control Plane
- BGP ASN assignments
- BGP import policy
- BGP export policy
- BGP peer group design
- BGP next-hop self

VXLAN Configuration
- VTEPs
- VNIDs
- Routing instances
- BUM traffic

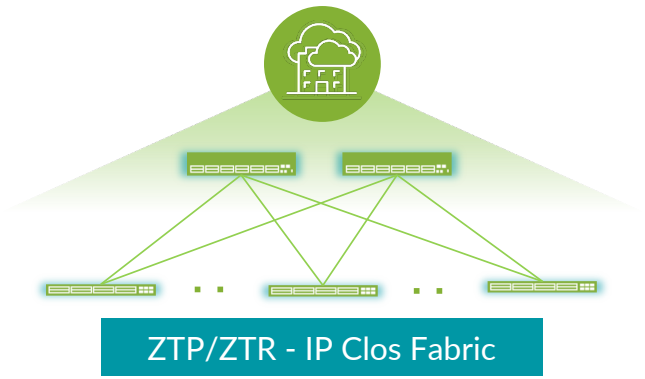
柔軟性やスケーラビリティは高いが多数の設定とパラメータが必要
管理性を高くするためは自動化がひとつの解となる

ジュニパー ファブリック ソリューション



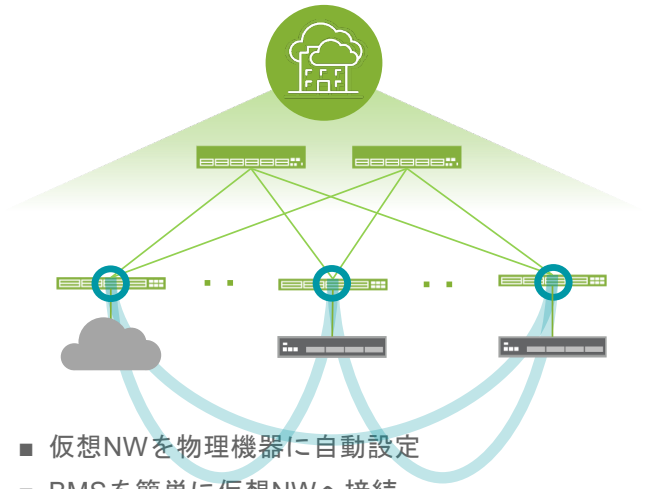
スケーラブルファブリックのライフサイクルを自動化

DAY-0 : セットアップ



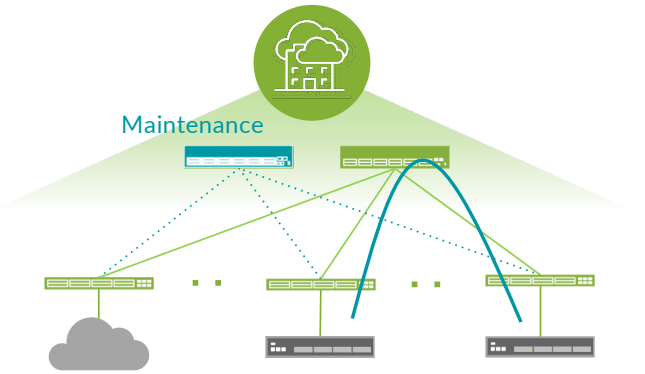
- Contrail Controllerからファブリック全体を管理
- 物理機器はラッキング、ケーブリングのみ実施
- ファブリックの設定は全て自動化
- Multi DC Fabric対応

DAY-1 : 運用



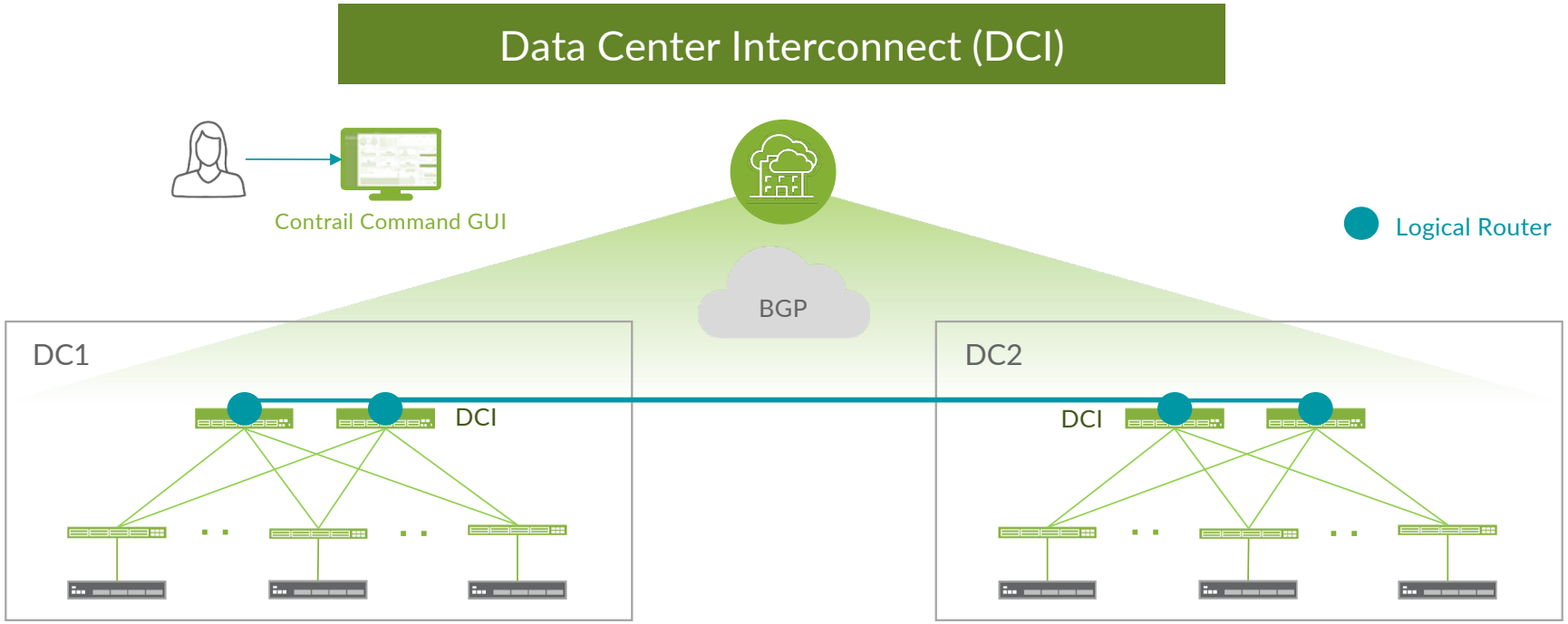
- 仮想NWを物理機器に自動設定
- BMSを簡単に仮想NWへ接続
- GW Routerで仮想NWを終端し外部接続
- PNFを使用したServiceChaining

DAY-2 : メンテンス



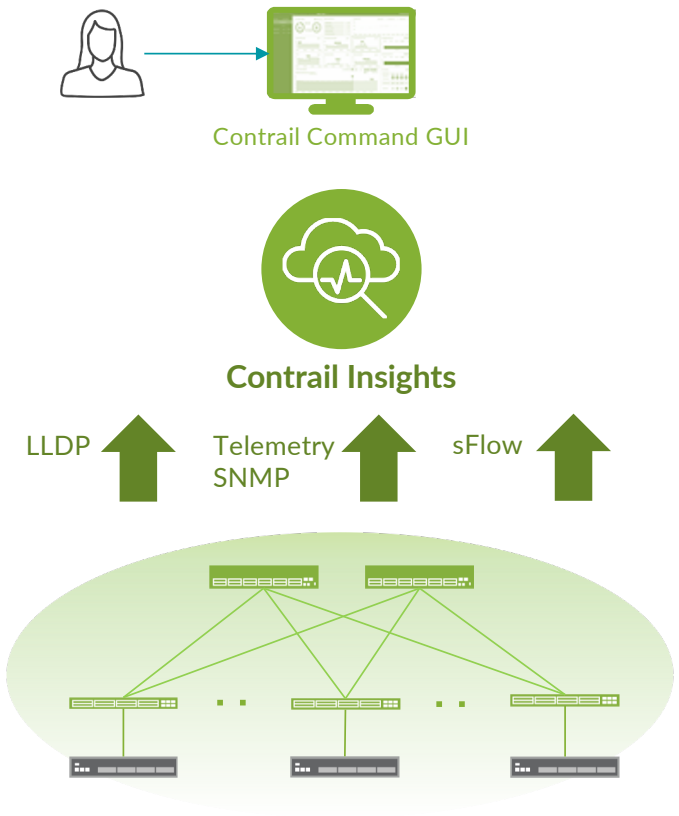
- メンテナンスモードによりトラヒックを自動迂回
- ファブリック全体を ヒットレスアップグレード
- 容易なスケールアウト
- Configを維持した容易な機器交換

CONTRAIL FABRIC OVERVIEW



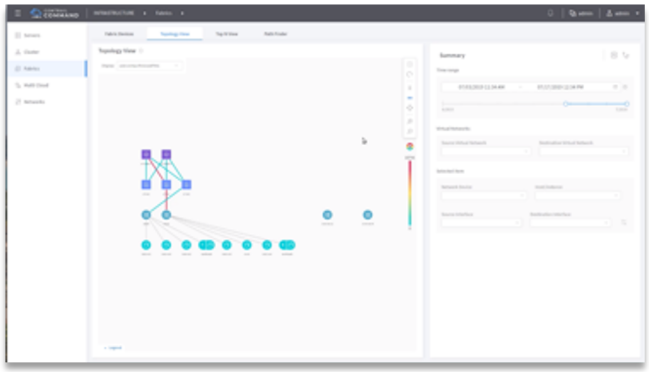
- データセンターのDCIデバイス間でBGPピアリング
- EVPN/VXLANベース
- ロジカルルータがそれぞれのテナント間の経路情報を交換
- シングルCEMクラスターで制御

CONTRAIL INSIGHTS (ファブリックの可視化) OVERVIEW



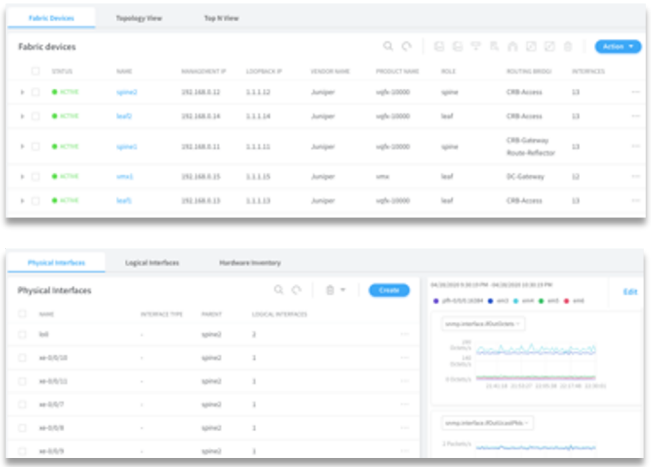
トポロジーヒートマップ

- ネットワーク、サーバ、ワークロード
- スマートトポロジーマップ
- 様々なメトリックでのリソースの可視化
  (ヒートマップ)



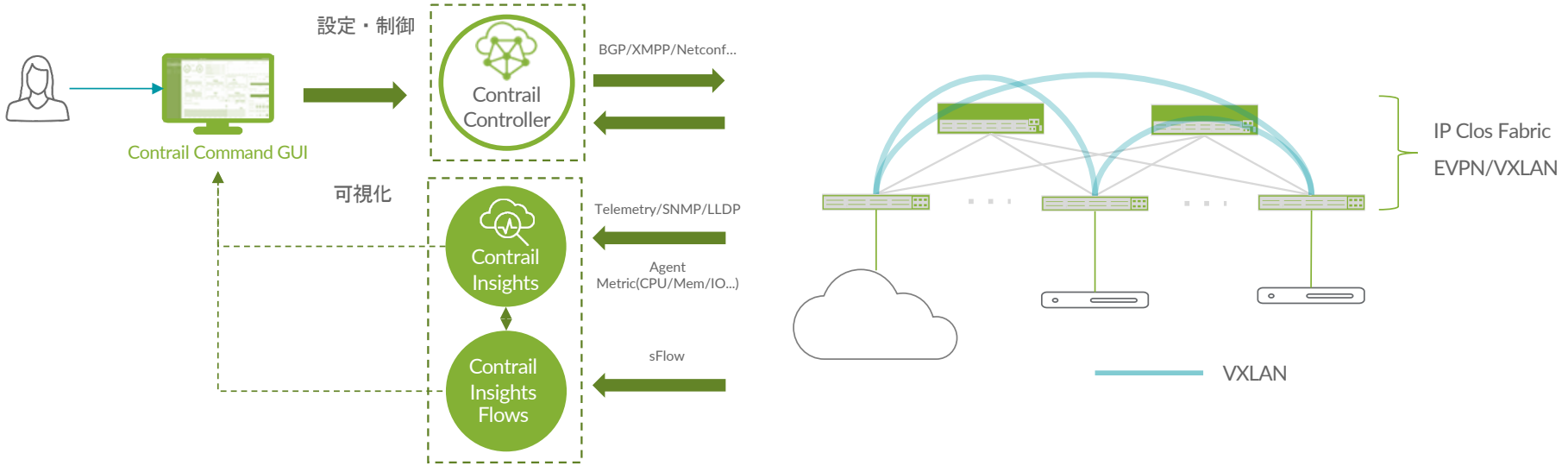
ファブリックデバイス一覧

- デバイスの一覧
  (Device種別、Version, Serial, IP, Role, etc)
- インタフェース一覧
- ハードウェアインベントリ

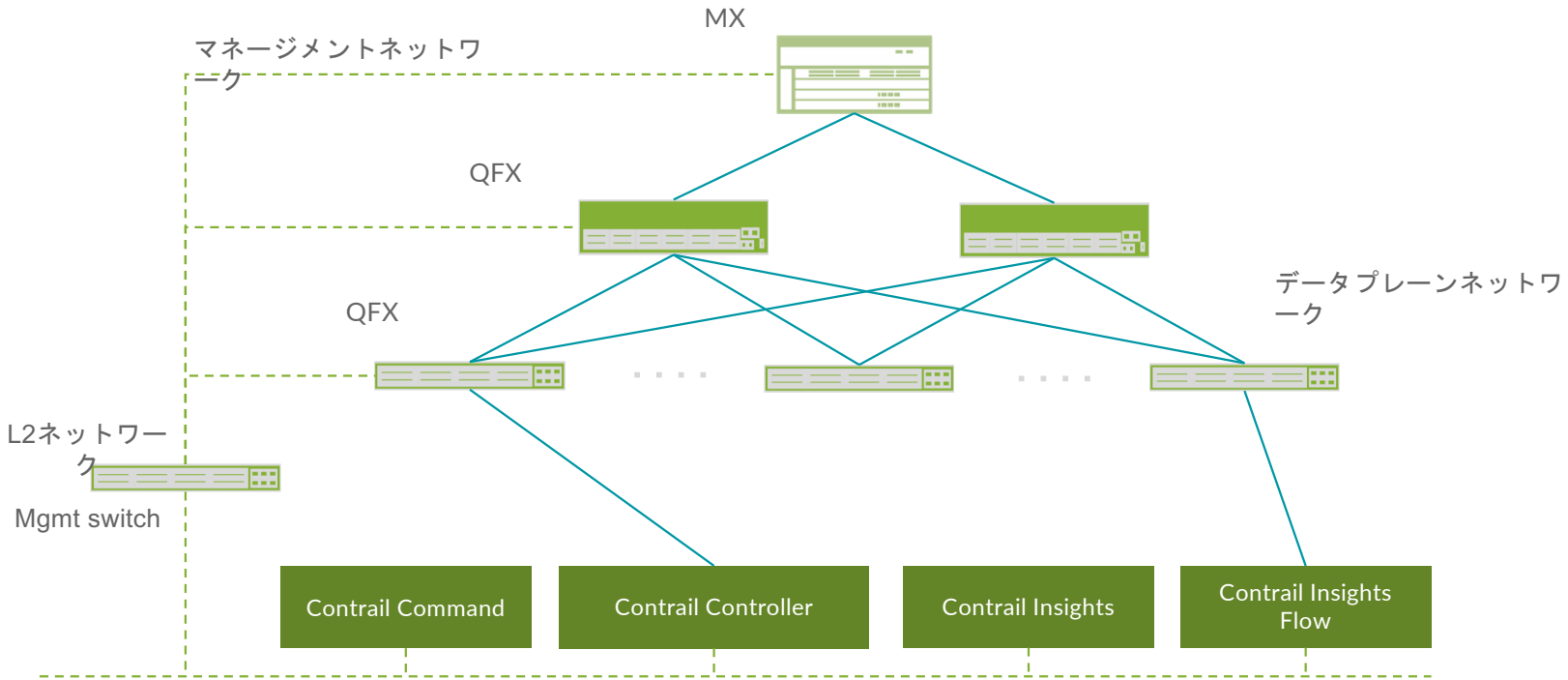


CONTRAIL FABRIC コンポーネント

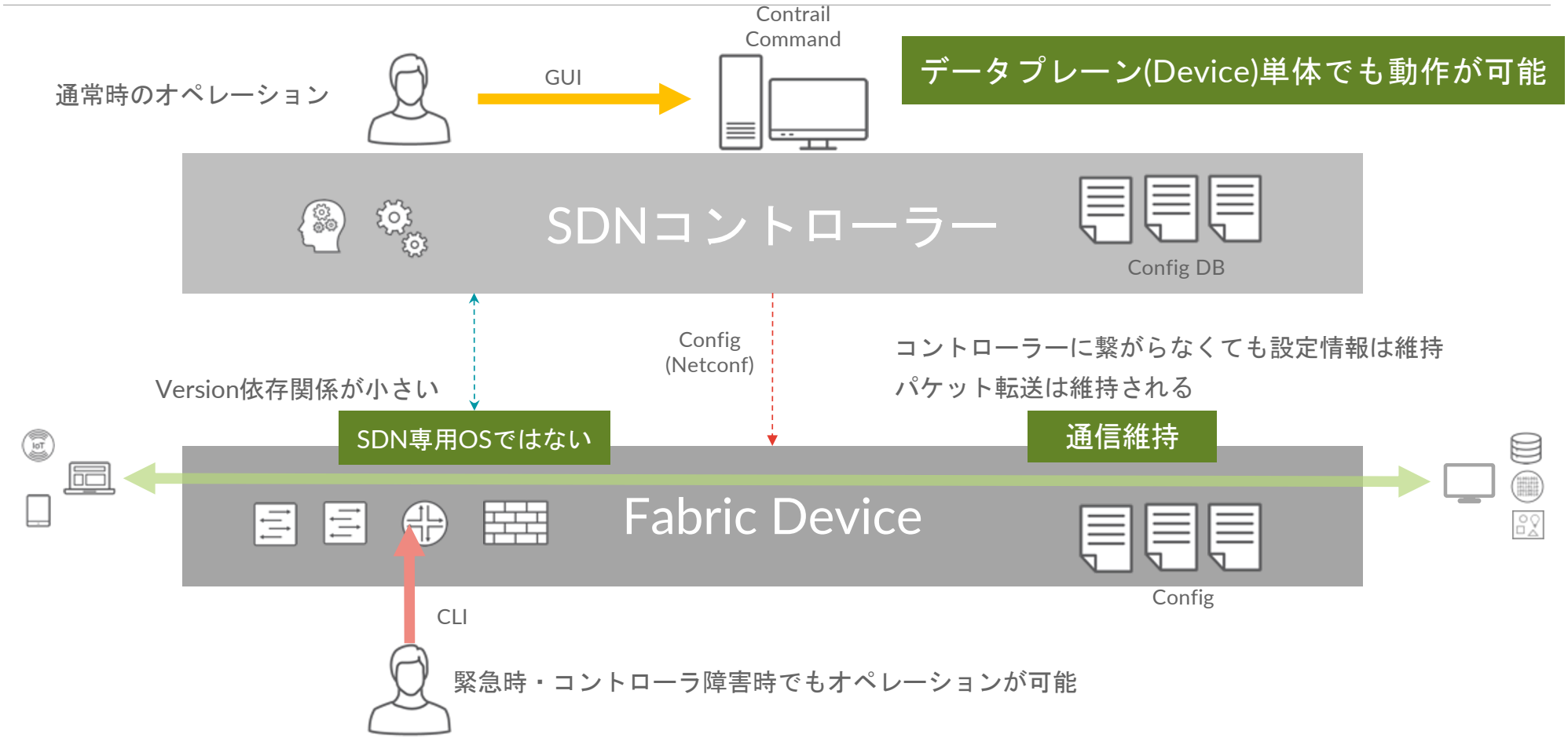
サーバ	主な機能
Contrail Command	Contrailのコンテナ、OpenStack等、GUI経由で提供します。
Contrail Controller	SDNのController機能を提供します。
Contrail insights	サーバ、ネットワークの可視化をGUI経由で提供します。
Contrail Insight Flows	サーバ、ネットワークの可視化を行う際のフローデータを収集します。



CONTRAIL FABRIC ネットワーク構成



疎結合SDNコントローラー



CEM構成例

記載以外の構成等、詳細は弊社営業・SEへお問い合わせください

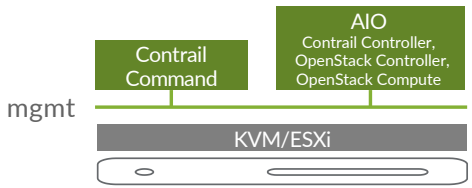
構成1 – CEM ALL-IN-ONE(AIO)

用途

CEMとOpenStackの簡易動作確認用

必要スペック

サーバ用途	サーバ１台あたり			台数	備考
	vCPU (HT)	メモリ（GB）	ストレージ（GB）		
Contrail Command	4	32	100	1	
AIO	20	48	350	1	



構成2 – CEM ALL-IN-ONE(AIO) + FABRIC

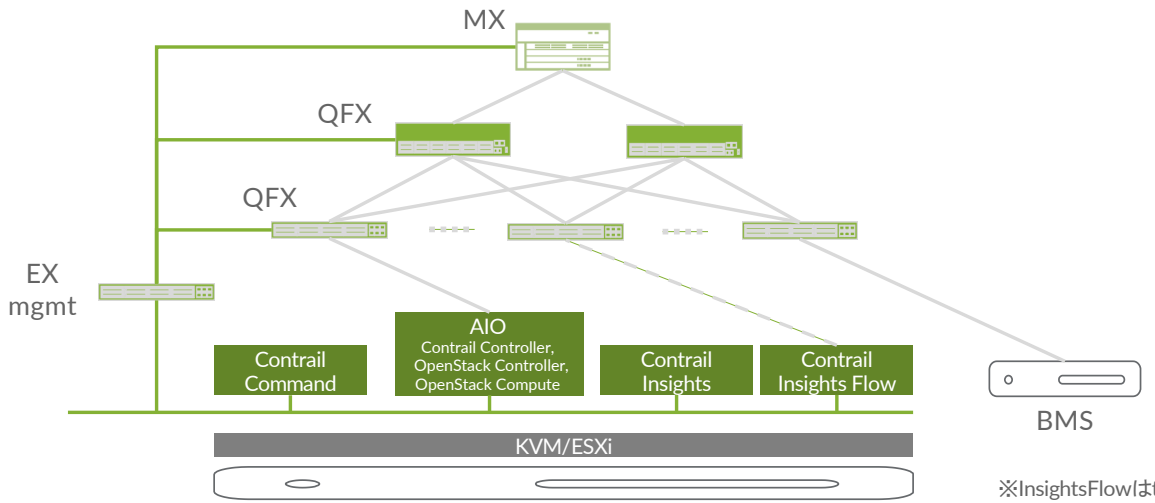
用途

CEMとOpenStackの簡易動作確認用

Fabric動作確認がメイン

必要スペック

サーバ用途	サーバ１台あたり			台数	備考
	vCPU (HT)	メモリ（GB）	ストレージ（GB）		
Contrail Command	4	32	100	1	
AIO	20(40)	48(96)	350	1	Max 32 Device (128 Device) Max 128K VMI(VPG*VLAN)
Contrail Insights	32	32	1000	1	Max 32 Device
Contrail Insights Flow	16	48	500	1	Max 64 Device / 100M Flow



※InsightsFlowはtelemetry InterfaceとしてOut-Band/In-Band 選択可能

構成3 – CEM + OPENSTACK(KOLLA)

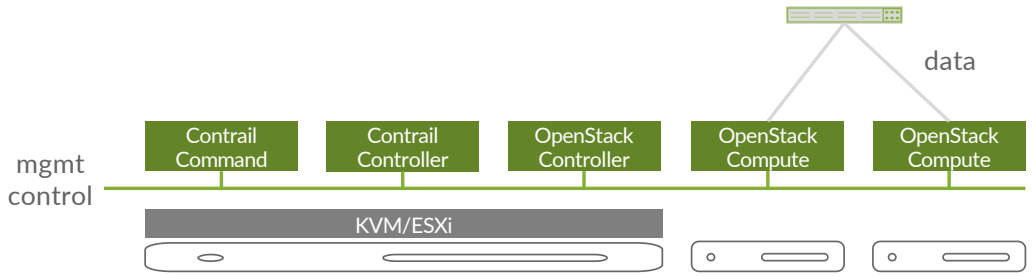
用途

CEMとOpenStackの動作確認用

必要スペック

サーバ用途	サーバ１台あたり			台数	備考
	vCPU (HT)	メモリ（GB）	ストレージ（GB）		
Contrail Command	4	32	100	1	
Contrail Controller	20	48	350	1	
OpenStack Controller	2	8	100	1	
OpenStack Compute	4+	8+	100+	2	VM数に依存

※OpenStack
https://docs.openstack.org/newton/install-guide-rdo/overview.html#figure-hwreqs



構成4 – CEM + OPENSTACK(KOLLA) + FABRIC

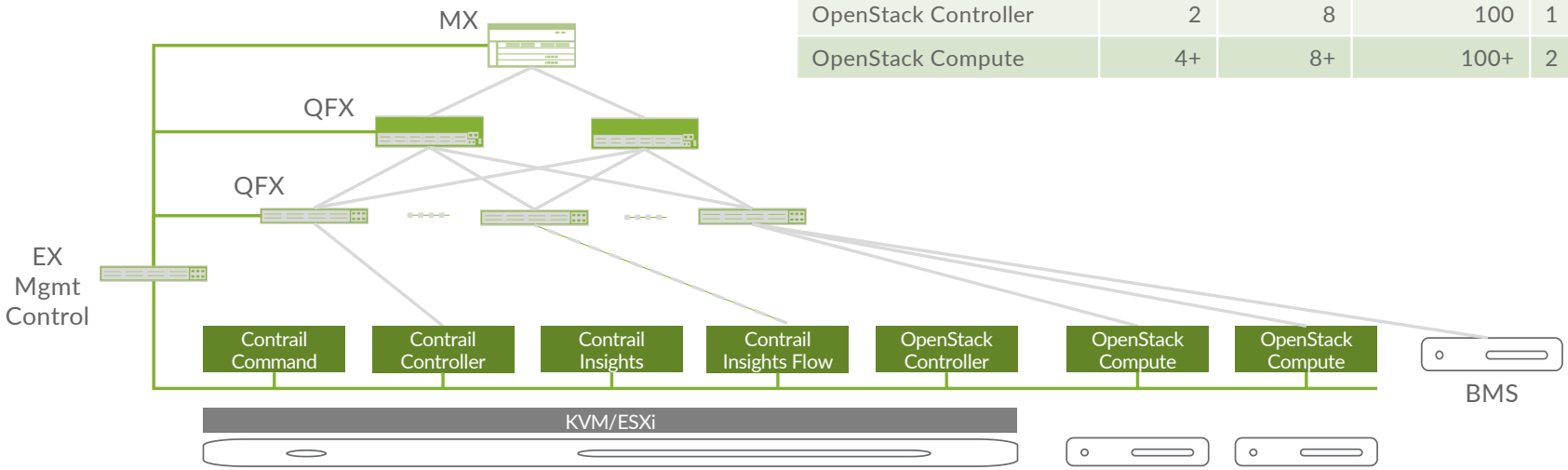
用途

CEMとOpenStackの動作確認用

OpenStack / BMS間接続、Internet接続

必要スペック

サーバ用途	サーバ１台あたり			台数	備考
	vCPU (HT)	メモリ（GB）	ストレージ（GB）		
Contrail Command	4	32	100	1	
Contrail Controller	20(40)	48(96)	350	1	Max 32 Device (128 Device) Max 128K VMI(VPG*VLAN)
Contrail Insights	32	32	1000	1	Max 32 Device
Contrail Insights Flow	16	48	500	1	Max 64 Device / 100M Flow
OpenStack Controller	2	8	100	1	
OpenStack Compute	4+	8+	100+	2	VM数に依存



※InsightsFlowはtelemetry InterfaceとしてOut-Band/In-Band 選択可能

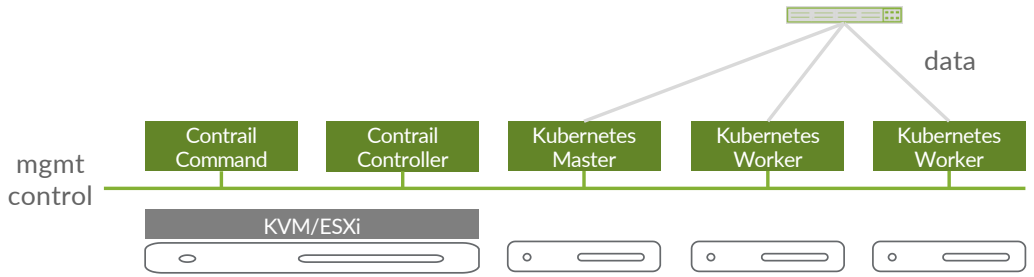
構成5 – CEM + KUBERNETES

用途

CEMとKubernetesの動作確認用

必要スペック

サーバ用途	サーバ１台あたり			台数	備考
	vCPU (HT)	メモリ（GB）	ストレージ（GB）		
Contrail Command	4	32	100	1	
Contrail Controller	20	48	350	1	
Kubernetes Master	1+	1+	20+	1	POD数に依存
Kubernetes Worker	1+	1+	20+	2	POD数に依存



構成6 – CEM + KUBERNETES + FABRIC

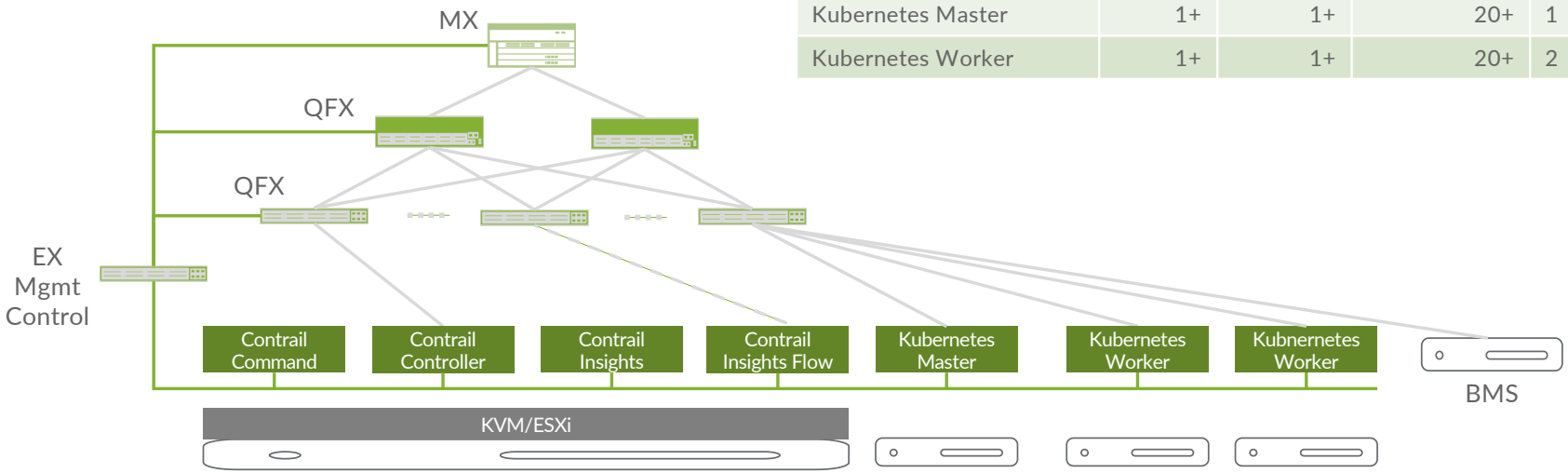
用途

CEMとKubernetesの動作確認用

Kubernetes / BMS間接続、Internet接続

必要スペック

サーバ用途	サーバ１台あたり			台数	備考
	vCPU (HT)	メモリ（GB）	ストレージ（GB）		
Contrail Command	4	32	100	1	
Contrail Controller	20(40)	48(96)	350	1	Max 32 Device (128 Device) Max 128K VMI(VPG*VLAN)
Contrail Insights	32	32	1000	1	Max 32 Device
Contrail Insights Flow	16	48	500	1	Max 64 Device / 100M Flow
Kubernetes Master	1+	1+	20+	1	POD数に依存
Kubernetes Worker	1+	1+	20+	2	POD数に依存



※InsightsFlowはtelemetry IntefaceとしてOut-Band/In-Band 選択可能

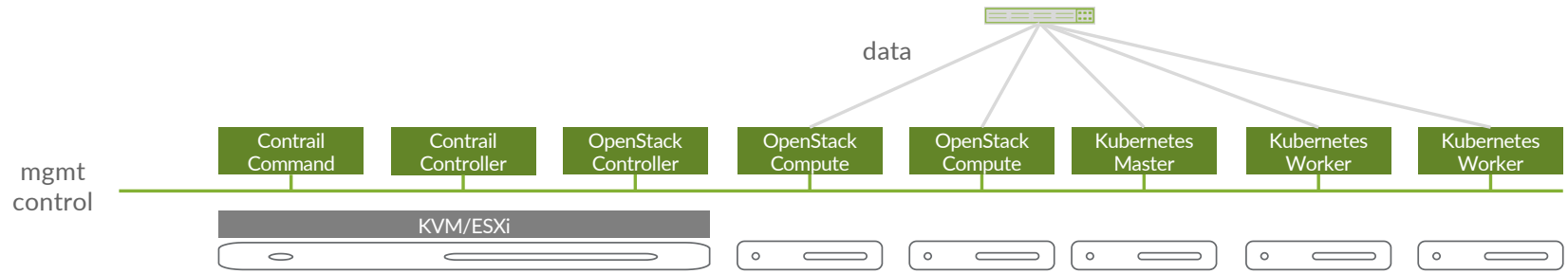
構成7 – CEM + OPENSTACK(KOLLA) + KUBERNETES

用途

CEMとOpenStack, Kubernetesの動作確認用

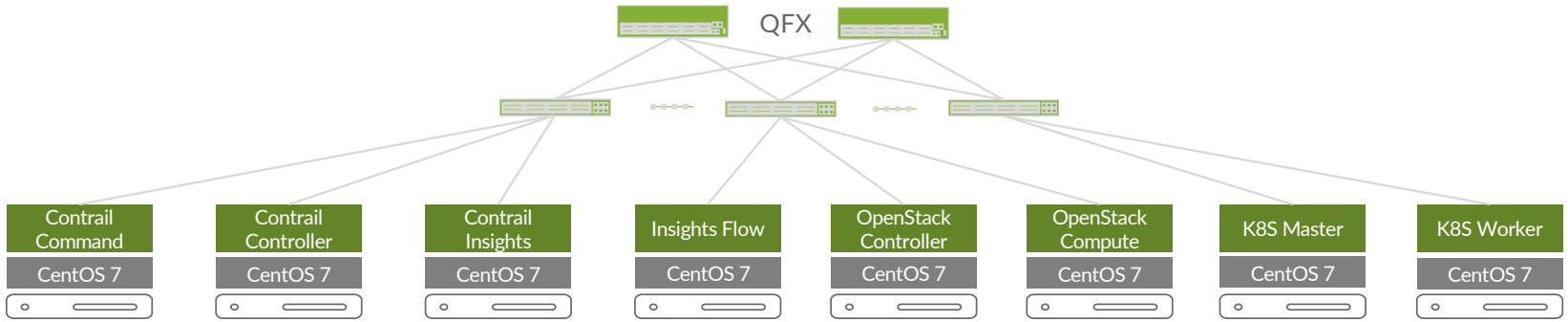
必要スペック

サーバ用途	サーバ１台あたり			台数	備考
	vCPU (HT)	メモリ（GB）	ストレージ（GB）		
Contrail Command	4	32	100	1	
Contrail Controller	20	48	350	1	
OpenStack Controller	2	8	100	1	
OpenStack Compute	4+	8+	100+	2	VM数に依存
Kubernetes Master	1+	1+	20+	1	POD数に依存
Kubernetes Worker	1+	1+	20+	2	POD数に依存



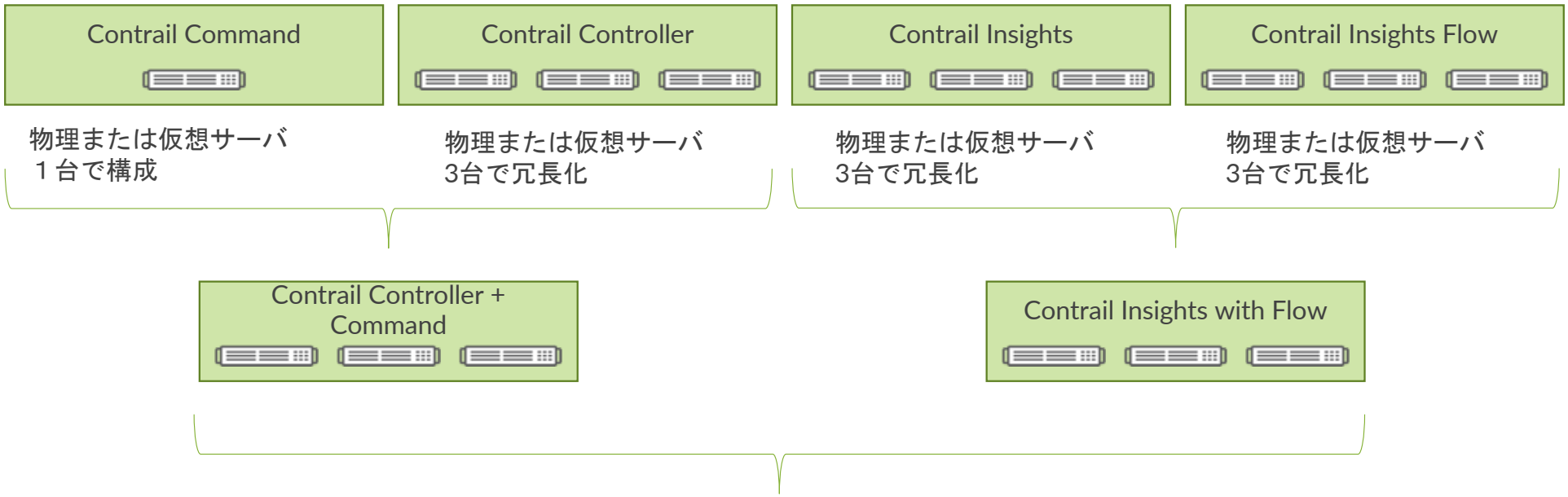
ALL-IN-ONE(AIO)

1. 各サーバ(VM可)にCentOSをインストール
2. Contrail Command用サーバにてDockerをインストールし、contrail-command-deployerをPull
3. Contrail-command-deployer内のAnsible playbookを使用し、Contrail Commandをインストール
4. Contrail Command GUIにLoginし、各サーバ(Contrail Controller, Contrail Insights, Insights Flow, OpenStack, K8S)をインストールするための設定を実施し、インストールを実行(BackgroundでAnsibleにより必要なコンポーネントが各サーバにインストールされる)



OpenStack/Kubernetesは含まれません

- サーバは用途別に配置し、物理サーバ利用が製品マニュアルの推奨です。（性能、拡張性の観点）
- サーバ機能を集約した場合、６台まで集約可能です。
- 仮想サーバ構成もサポートしています。
- 仮想化時にはコンピューティングノードとは別の環境に配置することが推奨です。
- Non-Mission–Critical環境（単純なFabric Management等）の際には他の仮想環境と同居は可能です。



CEM環境検討時には以下のURLを参照ください。

- CONTRAIL ENTERPRISE MULTICLOUD Datasheet
  ※ 4ページ目に「Minimum System Recommendations and Operating Environment」記載があります。
  *maxium

- Specifications Minimum System Recommendations and Operating Environment
  https://www.juniper.net/assets/uk/en/local/pdf/datasheets/1000637-en.pdf

- Contrail Networking Supported Platforms List
  https://www.juniper.net/documentation/en_US/release-independent/contrail/topics/reference/contrail-supported-platforms.pdf

- Installing Contrail Command
  https://www.juniper.net/documentation/en_US/contrail19/topics/example/install-contrail-command.html

- Appformix(Contrail insight)
- https://www.juniper.net/documentation/en_US/appformix/topics/concept/appformix-general-requirements.html

【YouTube】
- Using Contrail Command to Install Contrail Networking 1911 and Contrail Insights
  https://www.youtube.com/watch?v=nWxLMwS6ogY

CEM Release Policy

CONTRAIL SUPPORT POLICY

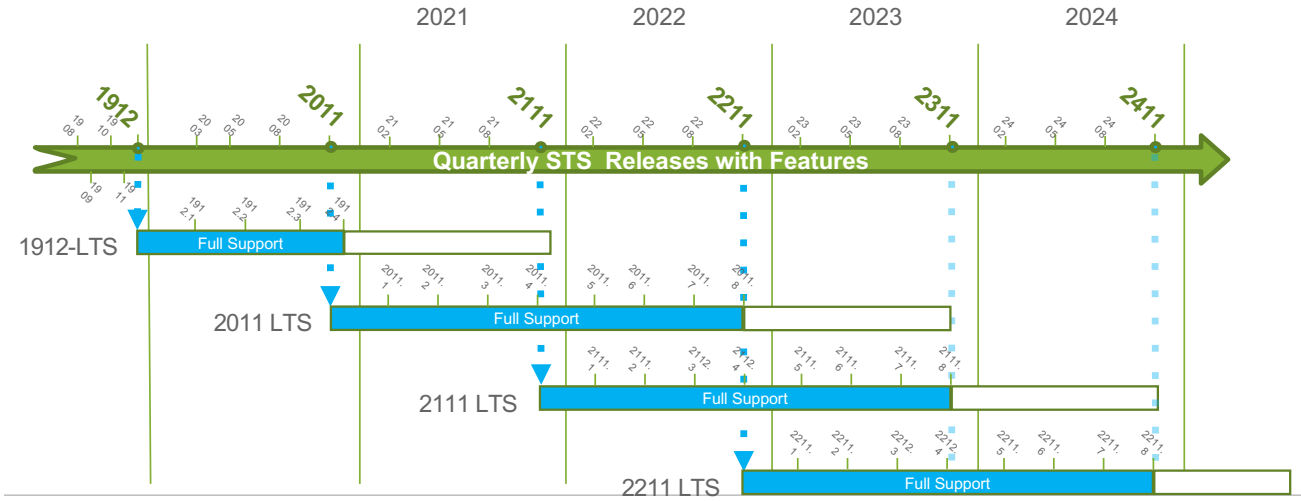
品質・ユーザビリティにフォーカスした製品を提供します

LTS
Long Term Support

- 3年間のLTS付きのリリースを毎年1回行います。（11月にリリースを原則）
- 3ヶ月毎にメンテナンスリリースを行います。
- バグ改修、セキュリティパッチ（最初の2年間）を対象とし、新機能の追加はありません。
- JTACでは3年間のサポートを提供します。

STS
Sort Term Support

- 3ヶ月毎に新機能を取り込んだSTSをリリースします。
- GA（General availability）後2年間のJTACサポートを提供します。



[その他のプロダクト情報]
・Contrail Insights は Contrail Fabric に従ってサポート提供されます。

※1912-LTSのみLTSの最初のリリースのため
Full Support 1年になっています

最後に

今までは同じ場所にあったクラウド・ワークロードのロケーション・ダイバーシティが進んでいる中、ネットワークによるクラウド間のコネクティビティ、セキュリティ・ガバナンスの複雑さが増してきています。ジュニパーはこれらの課題を根本的に解決します。

ジュニパーネットワークスはネットワーク・セキュリティ総合ベンダーだからこそ実現できるオープン・クラウド・エコシステムを目指します

- VUCA時代には**ビジネス・アジリティ**が最大の武器
- **オープン・エコシステム**を採用することで常に最適なサービス提供が可能に


Thank you
JUNIPER NETWORKS
Engineering
Simplicity