

Cisco DNA Programmability – ACI編 – 内容について

2021/8/30

Top Out Human Capital株式会社



Cisco DNA Programmability – ACI編 – (1日目)

9:30-9:40	オープニング	
9:40-12:00		REST,Pythonの基礎と概要の復習と演習の事前準備
12:00-13:00	昼休み	
13:00-16:00	ACI プログラマビリティ	ACIの理解
		ACIプログラマビリティオプション
		簡単な方法でACIスクリプトを作成する-WebArya
16:00-17:00	ACIプログラム応用	Cobraを使用してACI Pythonスクリプトに「Bite」を追加
		Cobraを使用してアプリケーションヘルスダッシュボードを構築

Cisco DNA Programmability – ACI編 – (2日目)

9:30-10:10	ACIプログラム応用	Cobraを使用してACI Pythonスクリプトに「Bite」を追加
		Cobraを使用してアプリケーションヘルスダッシュボードを構築
10:10-12:10	ACIとAnsibleの紹介	AnsibleでCisco ACIの自動化を実行
		AnsibleでApplicationとNetwork Policy
12:10-13:10	昼食	
		AnsibleでACI as Codeとしてまとめる
13:10-15:00	ACIとTerraformの紹介	TerraformでCisco ACIの自動化を実行
		TerraformでApplicationとNetwork Policy
		ACI TerraformリソースでACI REST APIをラップする

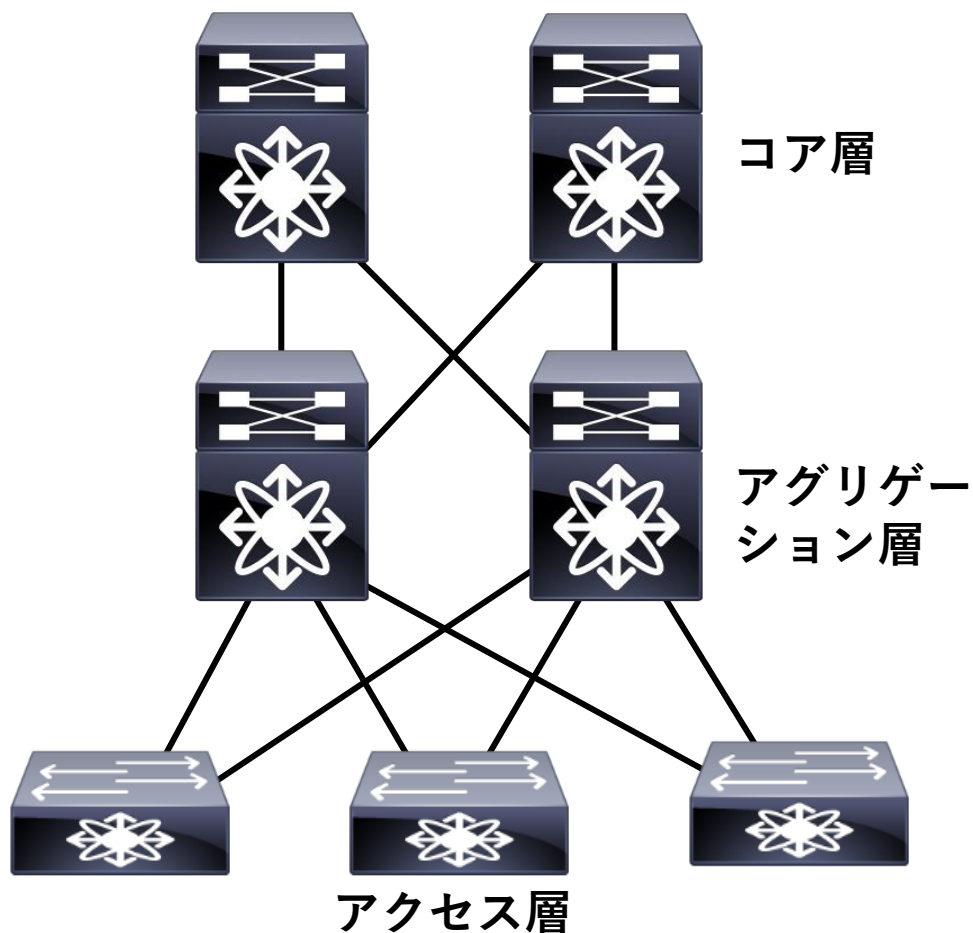
コースの日程、内容は進捗に応じて変更になる場合もありますのでご了承ください

前提条件

- ◆ Python、REST、JSON、POSTMAN、VRFについて理解していること
 - Cisco DNA Programmability Trainingをご受講し、ご理解いただいているようであれば、ほぼ満たしていると思われます。

ACI

従来のネットワークにおける問題点

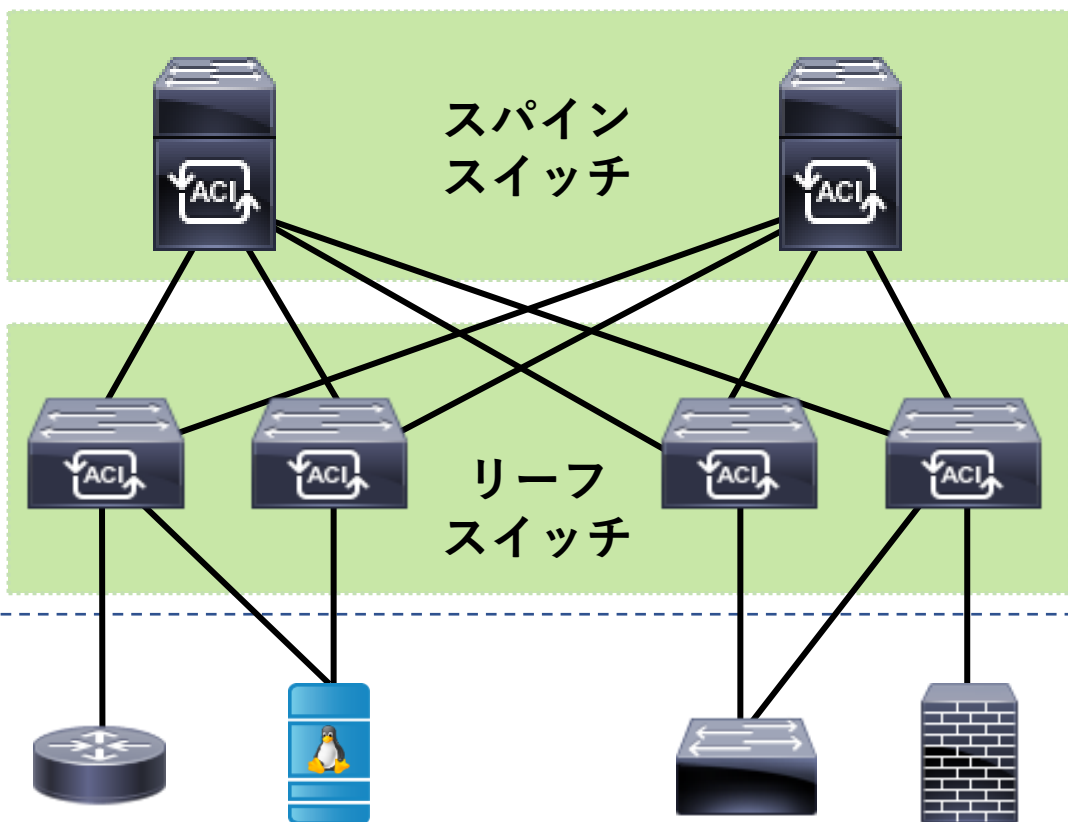


- ◆ 新規構築/設定変更
 - 手動設定
 - 作業に時間がかかる
 - 仮想化による複雑さの増大
 - 拡張性の限界
 - STPによるブロッキングポートの発生
- ◆ 管理性
 - デバイスごとに個別管理
 - 大規模化による管理負荷の増大
 - ネットワークチームとサーバチームの連携が必要
- ◆ セキュリティ
 - デフォルトですべてのトラフィックが許可

ACIによるネットワークの構築/運用/管理の改善



ACI ファブリック



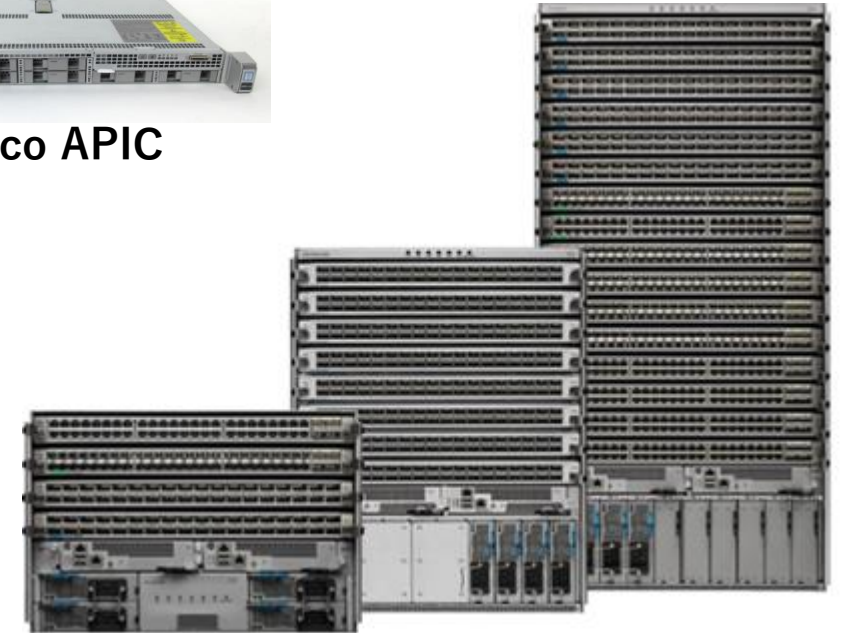
- ◆ 新規構築/設定変更
 - GUIを使用した設定
 - APIを使用した自動化
 - 作業時間の短縮、複雑さの軽減
 - 高い拡張性
 - STPの無効化 (IS-ISによるECMP)
- ◆ 管理性
 - 全てのデバイスを一元管理
 - 大規模環境も簡単に管理可能
 - ネットワークチームとサーバチームの作業範囲の明確化 (VMM連携)
- ◆ セキュリティ
 - ホワイトリストモデル

ACIの物理コンポーネント

- ◆ Cisco APIC
 - ACIファブリック全体を一元的に制御・管理するためのコントローラ
 - 3台以上でのクラスタ化を推奨
- ◆ Cisco Nexus 9500
 - スパインスイッチ
 - ACI対応ラインカードが必要
- ◆ Cisco Nexus 9300
 - スパインスイッチ or リーフスイッチ
 - ※ モデルにより異なる



Cisco APIC

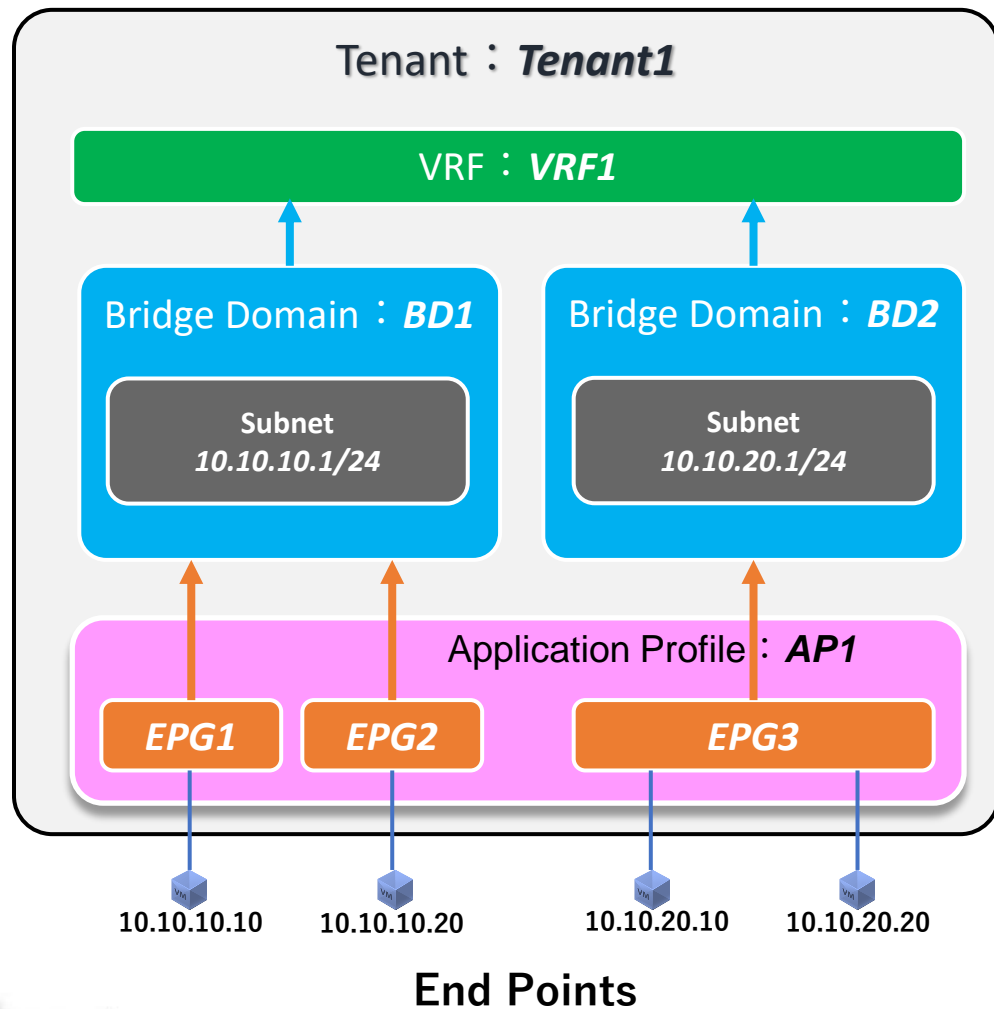


Cisco Nexus 9500



Cisco Nexus 9300

ACIの論理コンポーネント（オーバーレイ）



- ◆ Tenant
 - ACIファブリックを論理的に分割
- ◆ VRF (Virtual Routing and Forwarding)
 - テナント内を論理的に分割するルーティングインスタンス
- ◆ Bridge Domain
 - VRFに接続するレイヤ2ネットワーク
 - サブネットを格納するコンテナ
- ◆ Subnet
 - VRFのIPアドレス（デフォルトゲートウェイ）
- ◆ Application Profile
 - EPGを格納するコンテナ
 - 1つのアプリケーションを構成するコンポーネントをグループ化
- ◆ EPG (End Point Group)
 - エンドポイント（物理サーバや仮想マシン）を格納するコンテナ
 - 同じポリシーを共有するサーバをグループ化

ACIのオーバーレイとアンダーレイ

◆ オーバーレイ

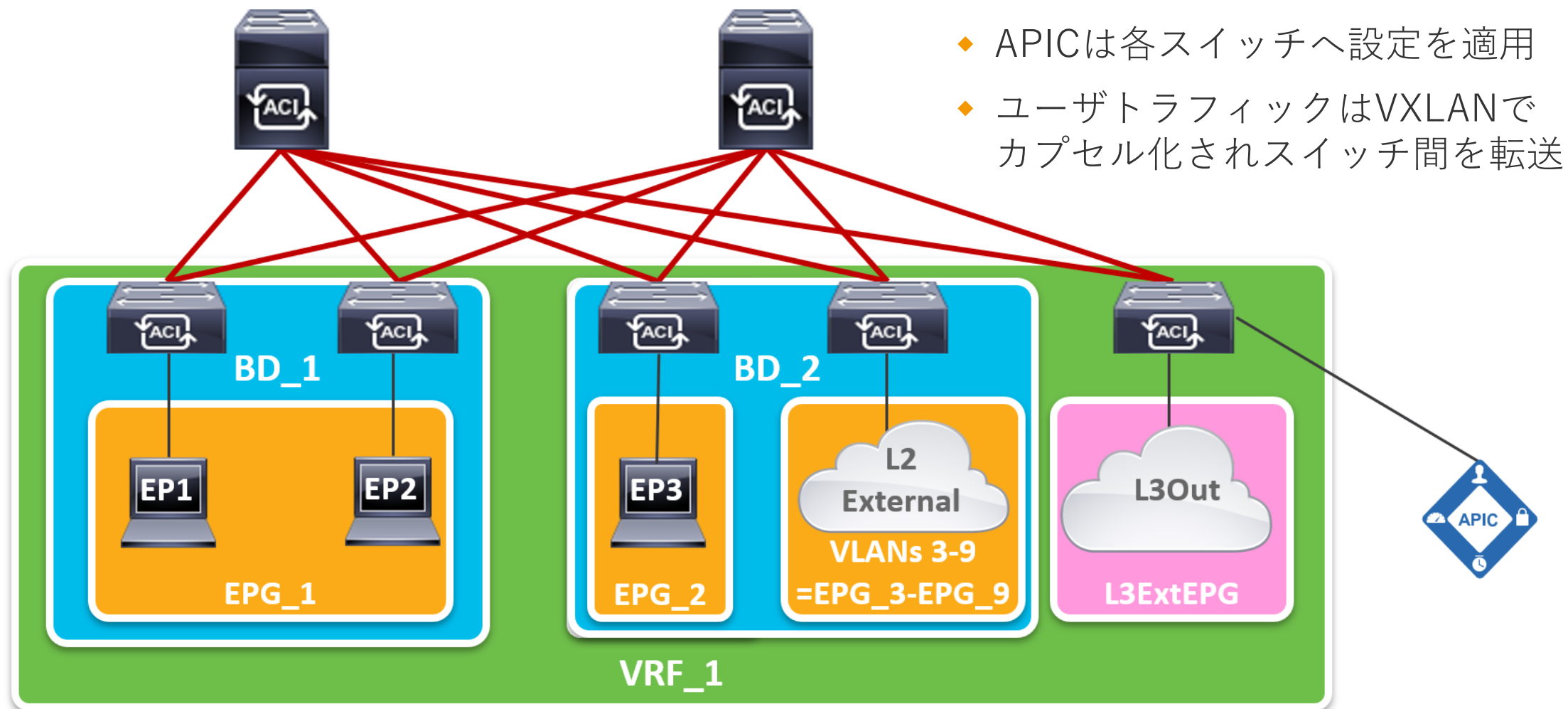
- 拡張バージョンのVXLANを使用（pcTagによる制御）
- ファブリック全体にまたがるレイヤ2スイッチングおよびレイヤ3ルーティング

◆ アンダーレイ





- 各スイッチ間の到達性を確保するためにIS-ISルーティングプロトコルを使用される
- 各スイッチ間のルートは等コストロードバランシングにより冗長化および負荷分散される

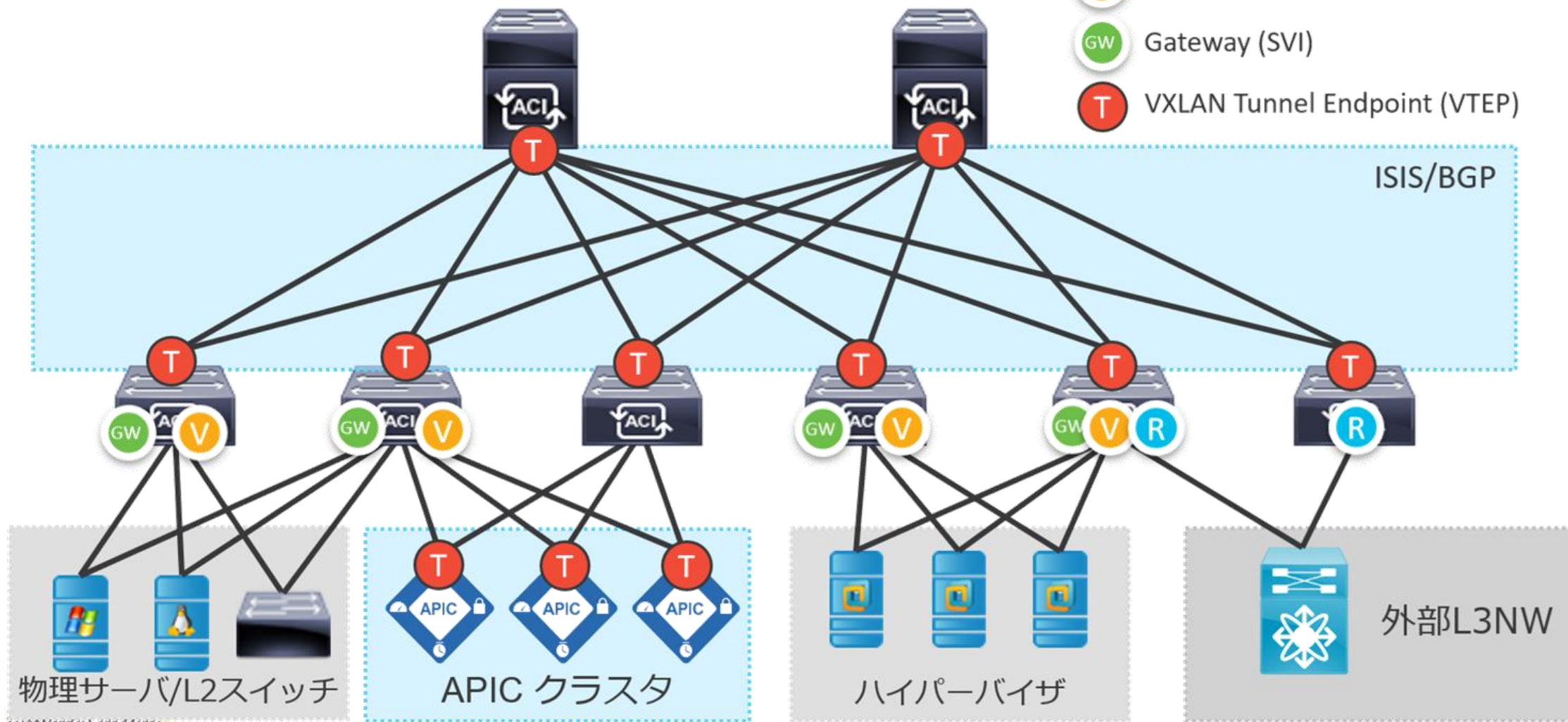
- ◆ APICに設定されたポリシーに従って、オーバーレイとアンダーレイの設定が自動的にNexusスイッチへプッシュされる
 - ユーザはオーバーレイとアンダーレイの詳細な仕組みを理解する必要はない

オーバーレイの動作イメージ



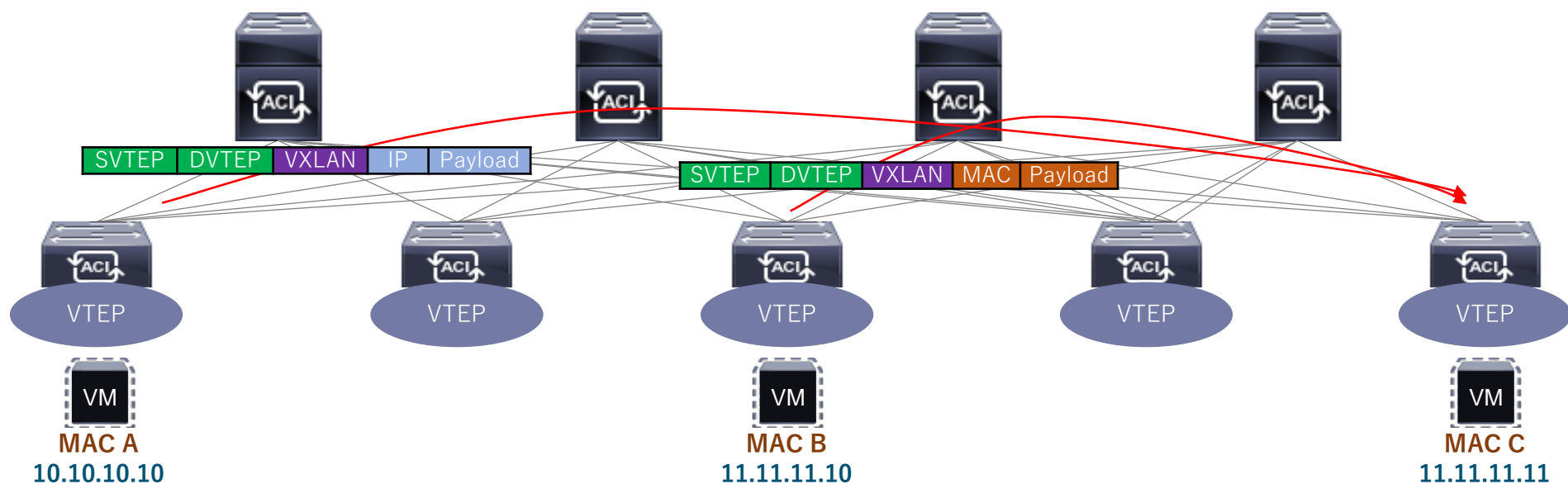
アンダーレイの動作イメージ

-  External Routing
-  VLAN
-  Gateway (SVI)
-  VXLAN Tunnel Endpoint (VTEP)



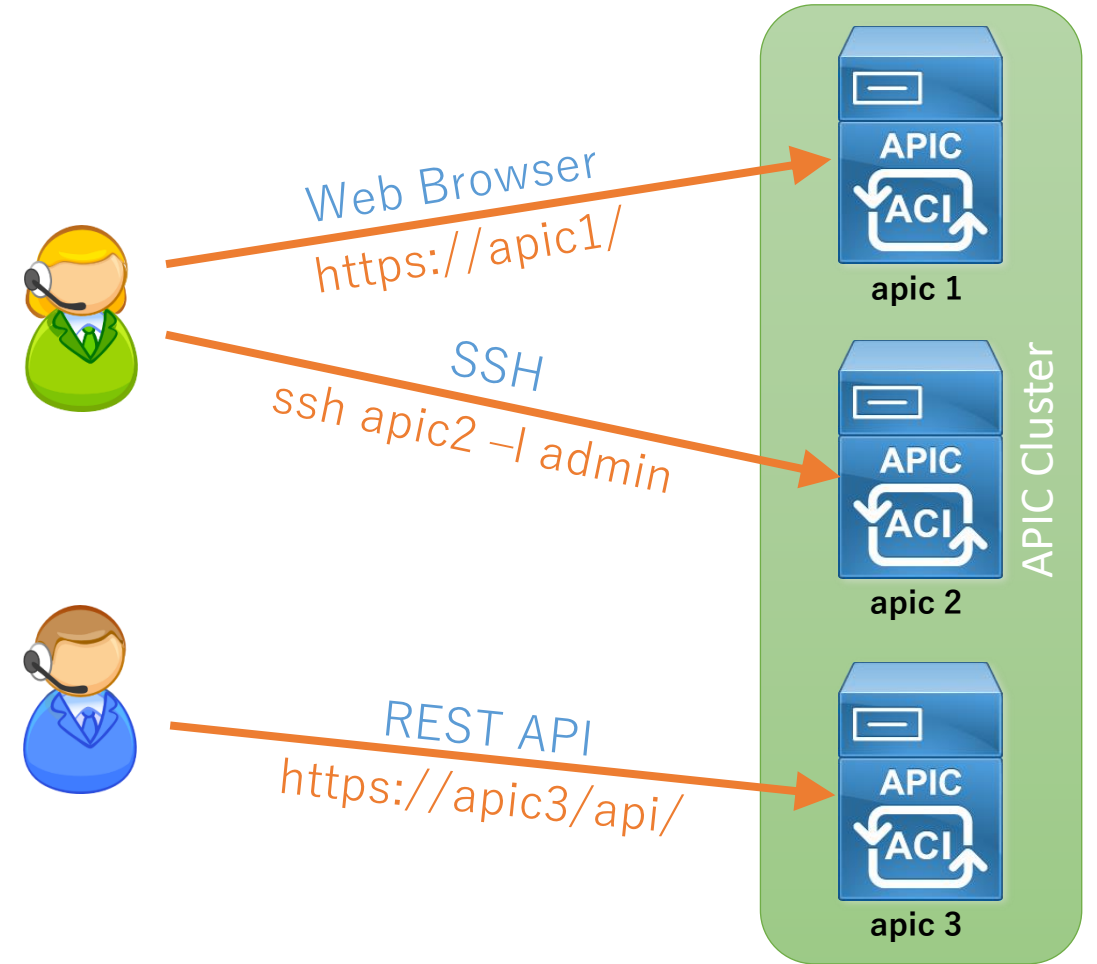
エンドポイント間のトラフィック転送

- ◆ レイヤ2トラフィック/レイヤ3トラフィック、いずれもVXLANでカプセル化されてACIファブリック内を転送される
 - ー 同じサブネット内（MAC B ⇒ MAC C）：レイヤ2転送
 - ー 異なるサブネット間（10.10.10.10 ⇒ 11.11.11.11）：レイヤ3転送




ACIの設定方法

- ◆ GUI
 - APICの管理IPにWebブラウザで接続
- ◆ CLI
 - APICの管理IPにSSHで接続
- ◆ API
 - APICの管理IP上で動作するREST APIを様々なツールから使用
 - Python
 - Ansible
 - Terraform
 - etc...



APICのGUI – システム ダッシュボード



System

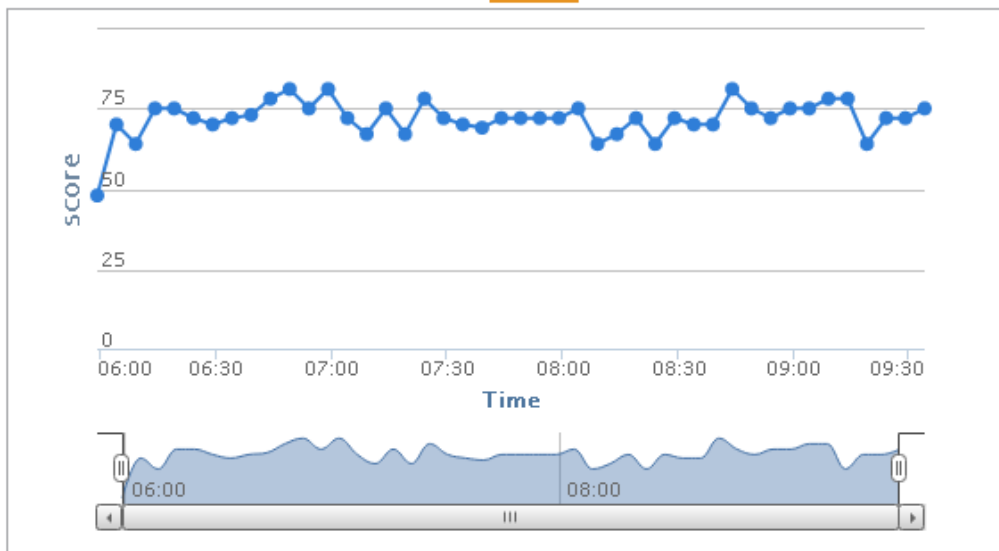
TenantsFabricVM NetworkingL4-L7 ServicesAdminOperations

QuickStart | Dashboard | Controllers | Faults | Config Zones

Advanced Mode
welcome, admin

System Health

81



Nodes With Health ≤ 99

99

Name	POD ID	Type	Health Score
leaf-1	1	leaf	90
leaf-2	1	leaf	90
spine-1	1	spine	90

Fault Counts By Domain

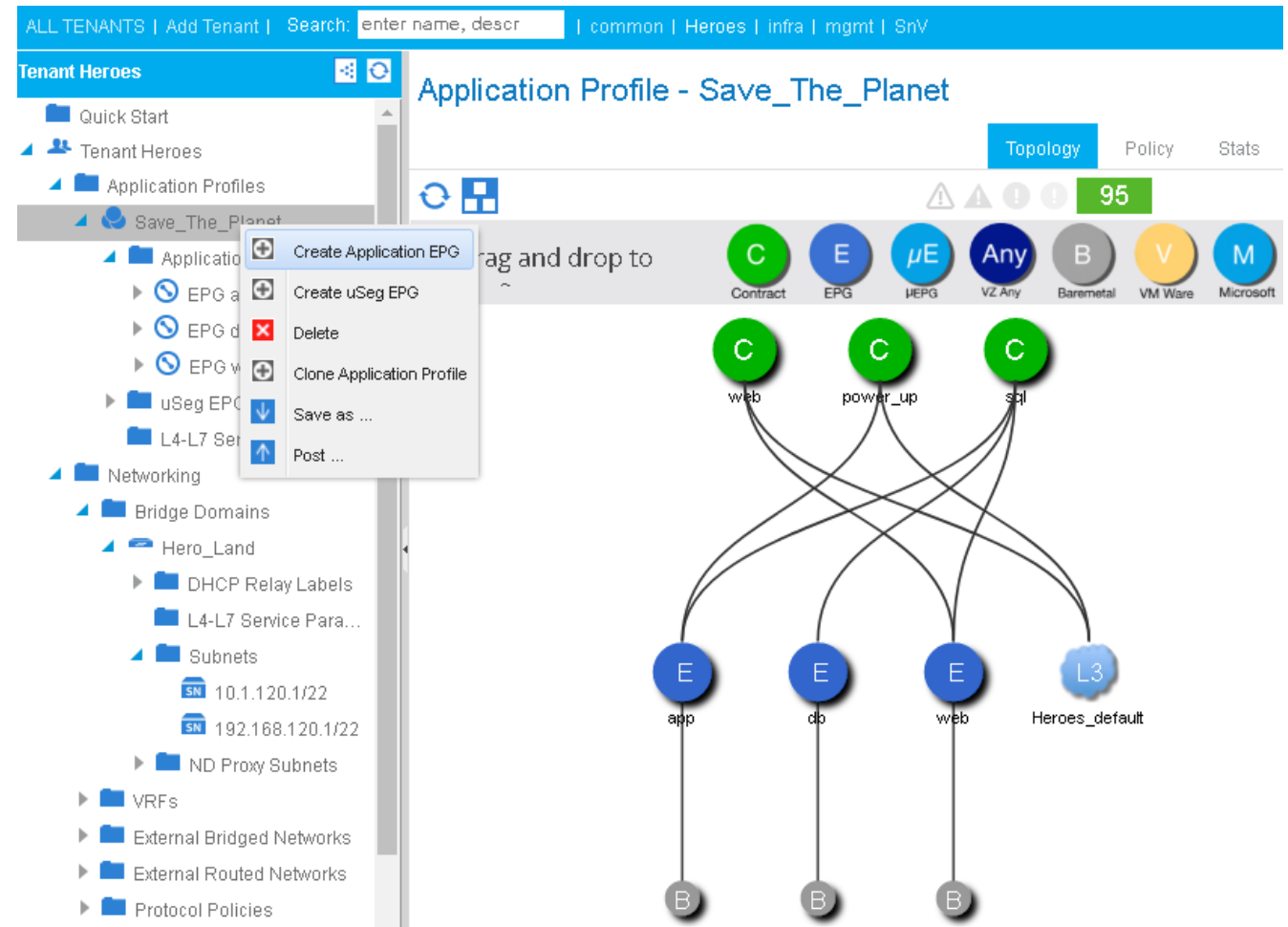
Fault Level:				
SYSTEM WIDE	3	4	9	441
Access	0	0	0	5
External	0	0	0	0
Framework	0	0	0	0
Infra	3	4	0	436
Management	0	0	0	0
Security	0	0	0	0
Tenant	0	0	9	0

Fault Counts By Type

Fault Level:				
Communicati...	0	0	0	0
Config	0	0	9	0
Environmental	0	3	0	0
Operational	3	1	0	441

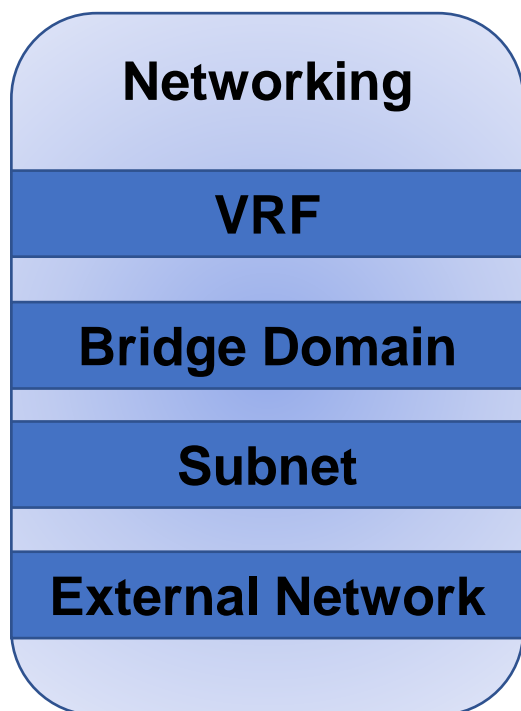
APICのGUI – テナント内の設定

- ◆ 設定メニューは階層化されたツリー構造
 - ― 例) EPGを作成する場合
 - Tenant > Application Profile > Application EPG
- ◆ Save as ...
 - ― 対象オブジェクトの設定パラメータをJSON / XMLフォーマットでダウンロードできる



オーバーレイの構成要素 – テナント

Tenant A



Policy

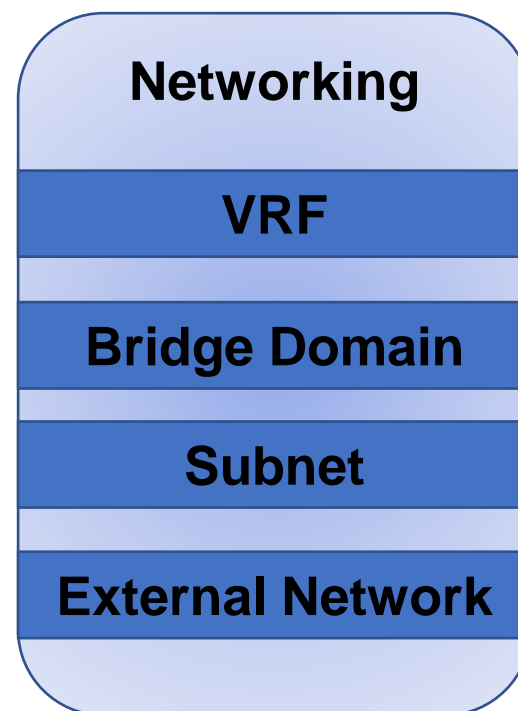
Application Profile

EPG

Contract

Filter

Tenant B



Policy

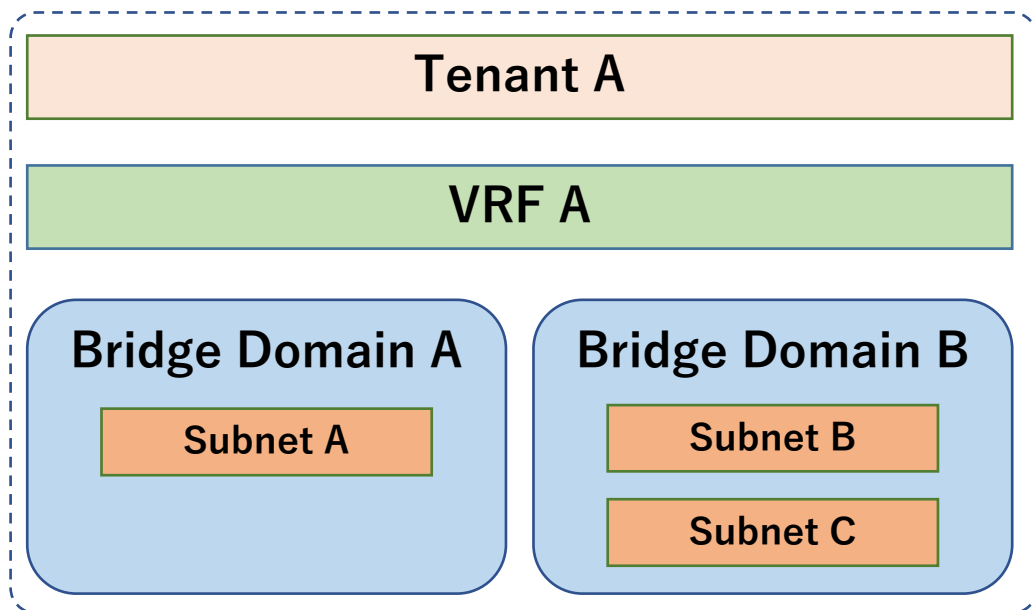
Application Profile

EPG

Contract

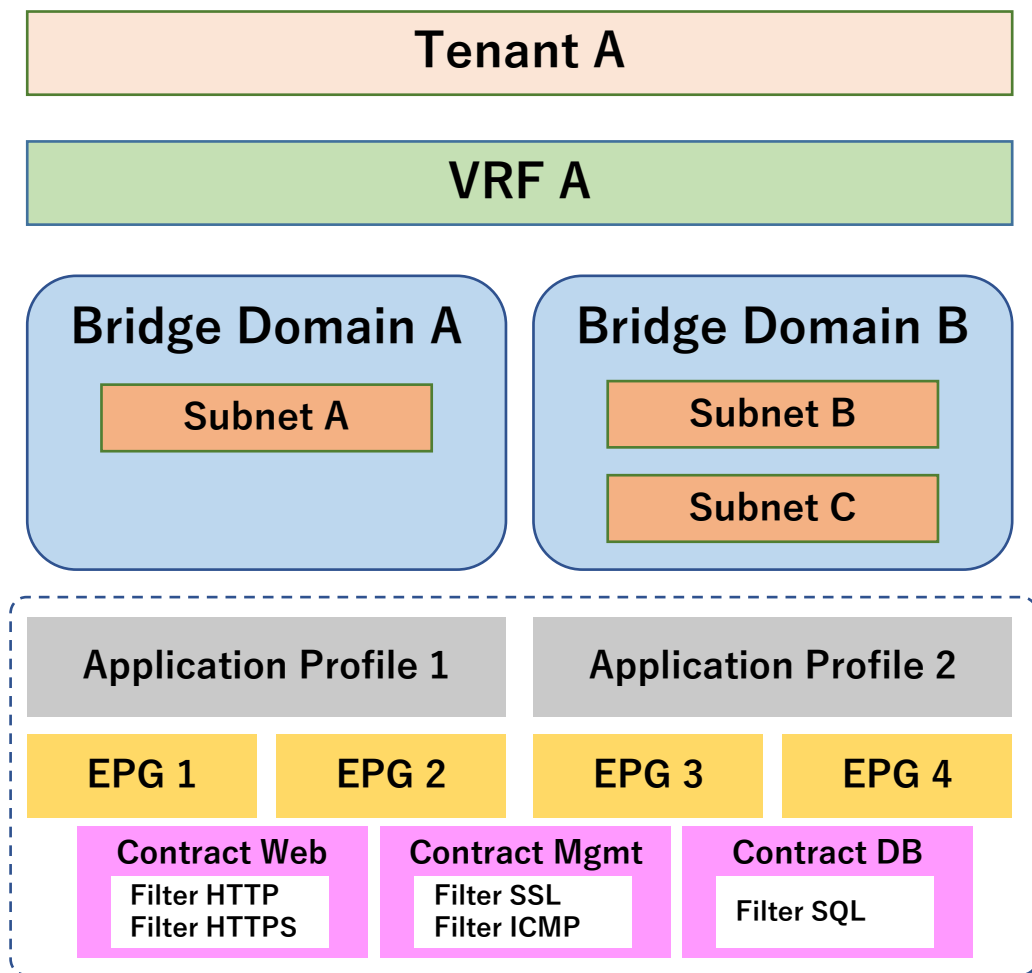
Filter

オーバーレイの構成要素 – テナントネットワーク



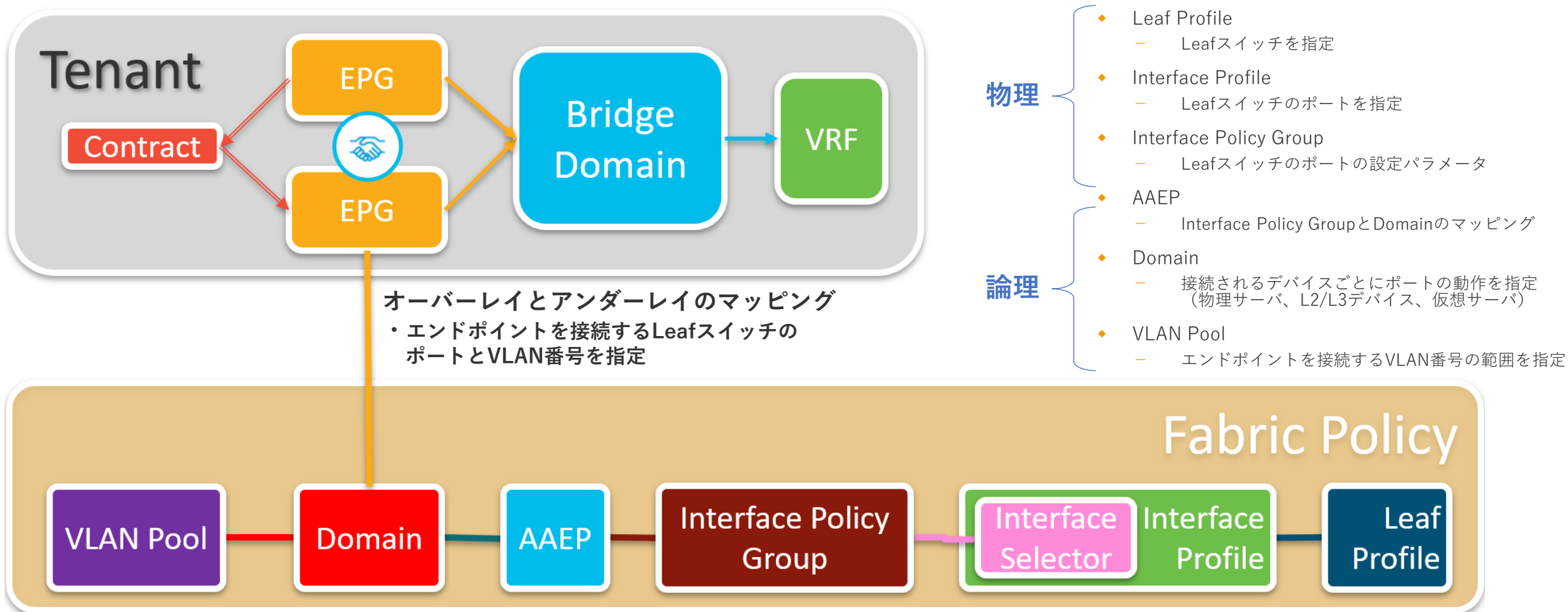
- ◆ ACIファブリックを複数のテナントに分割
 - Mgmt、Commonなどデフォルトで存在するテナントもある
- ◆ テナント内のネットワークをVRFで分割
 - VRF間の通信はデフォルトで不可
- ◆ VRF配下のネットワークをBridge Domainで分割
 - Bridge Domainごとに異なるL2NW（ブロードキャストドメイン）
 - ARPやUnknown UnicastをFloodingするかどうかはBridge Domainごとに設定可

オーバーレイの構成要素 – テナントポリシー



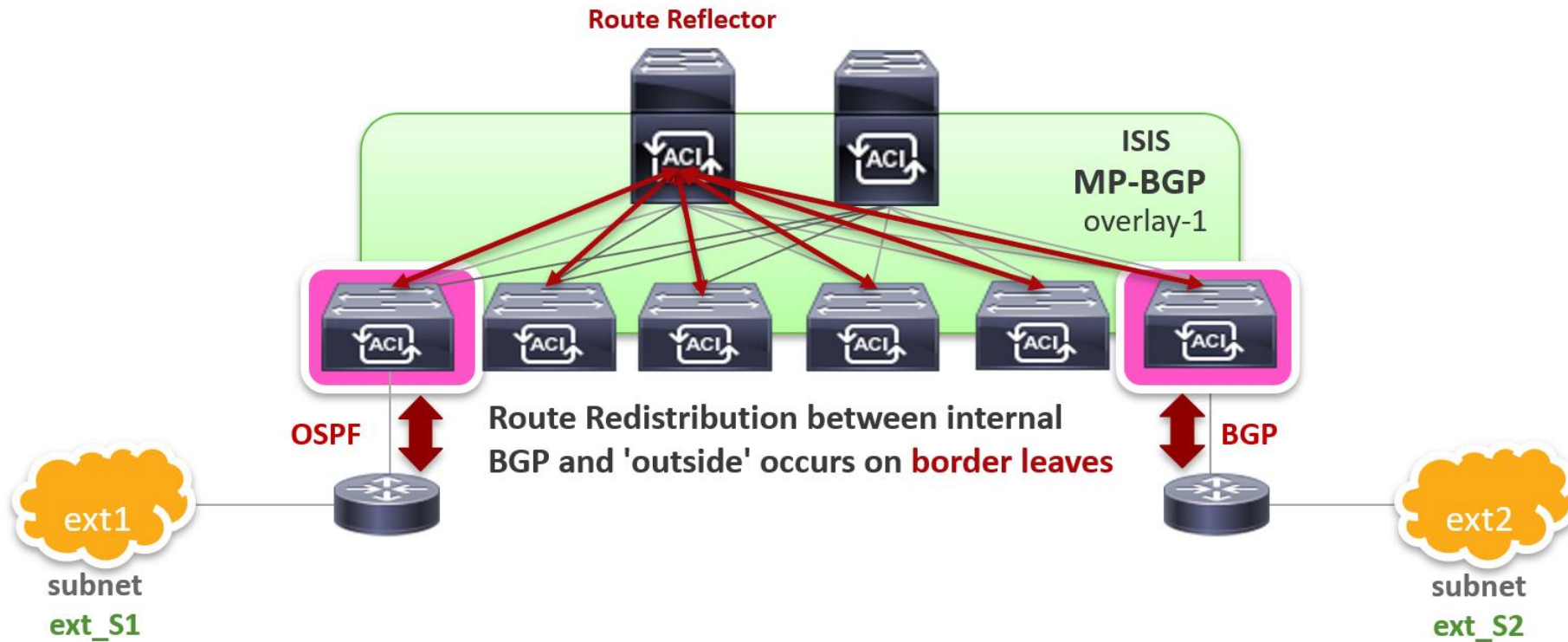
- ◆ Application ProfileでEPGをグループ化
- ◆ EPGでエンドポイントをグループ化
 - 同じEPGに所属するエンドポイント同士の通信はデフォルト許可
 - 異なるEPGに所属するエンドポイント同士の通信はデフォルト拒否
 - デフォルトを許可にすることも可能
- ◆ ContractでEPG間の通信を許可
 - Filterによって許可する通信を指定
 - PBRを実行して、ACIに接続された外部デバイスを経由させることも可能
 - ファイアウォール、IPSなど

(参考) アンダーレイの構成要素 – ファブリックポリシー



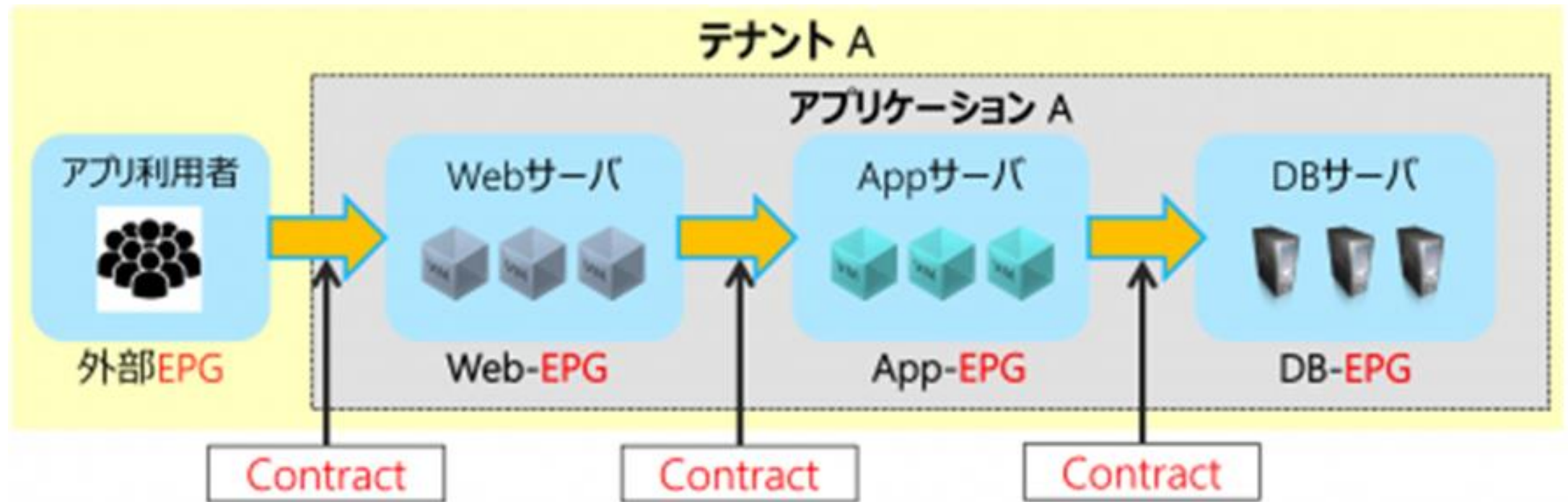
(参考) 外部レイヤ3接続

- ◆ ACIと外部L3デバイスを接続することが可能
 - スタティックルートまたはルーティングプロトコル（OSPF、EIGRP、BGP）を使用
 - ACI内部ではMP-BGPを使用して、各Leafスイッチへルートを通知



EPGとContract

- ◆ ACIにおいて定義された抽象的な「グループ」と「通信ルール」
- ◆ 物理・仮想の違い、接続場所違い、台数の増減などに影響されない



Contractにおける通信の制御

- ◆ Contractは「通信をどう制御するか」だけが含まれ、「どこ」は含まない

Contract

Contract Subject ※一つのContractに複数のContract Subjectを構成可能

Filter **Ether Type:** IP, MPLS Unicast, Trill, ARP, FCoE, MAC Security, Unspecified

IP: tcp, udp, icmp, icmpv5, Unspecified etc.

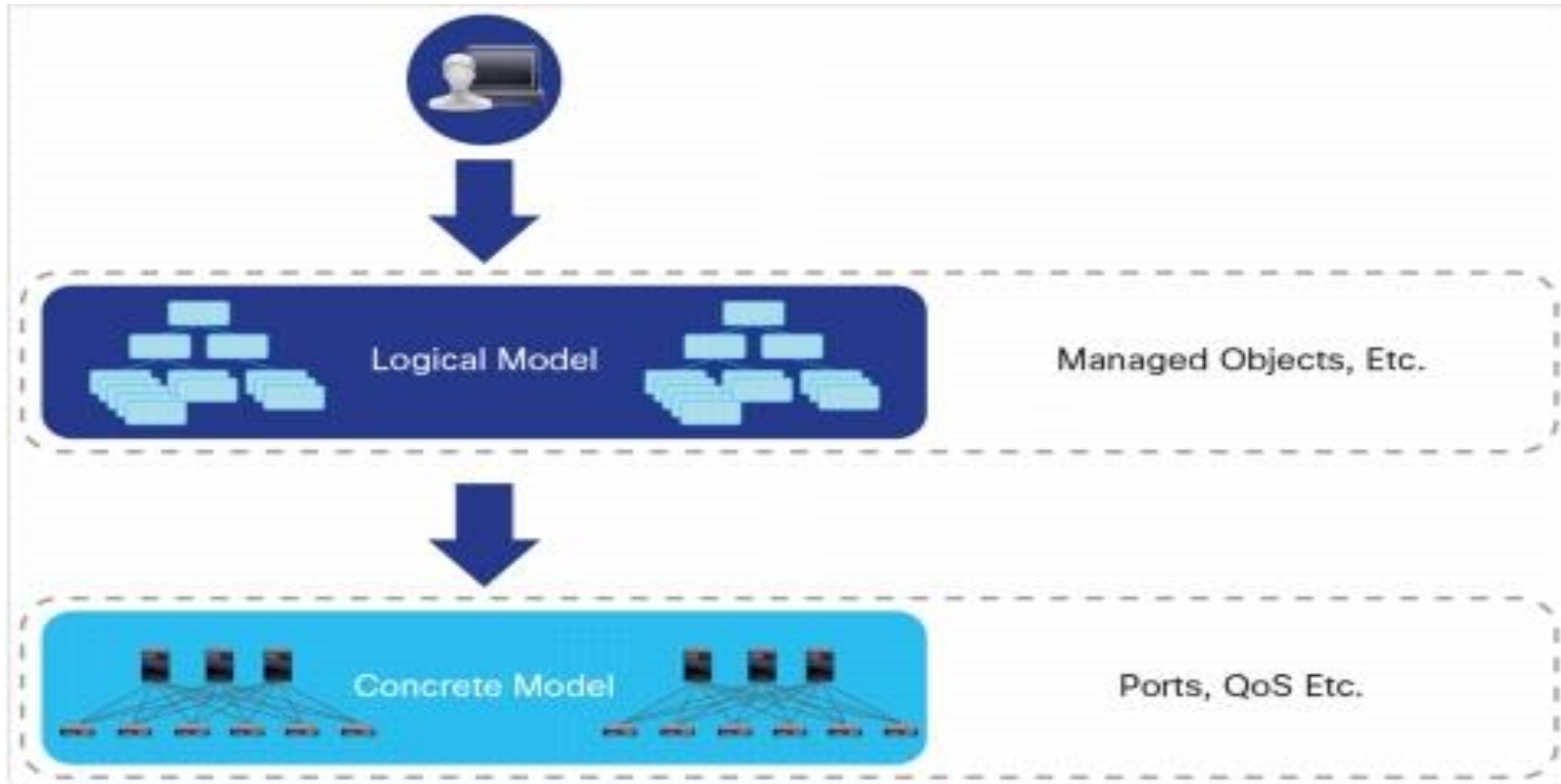
Port: Source Port / Destination Port (いずれもレンジ指定可能)

Service Graph (L4-L7連携)

QoS Class, Target DSCP

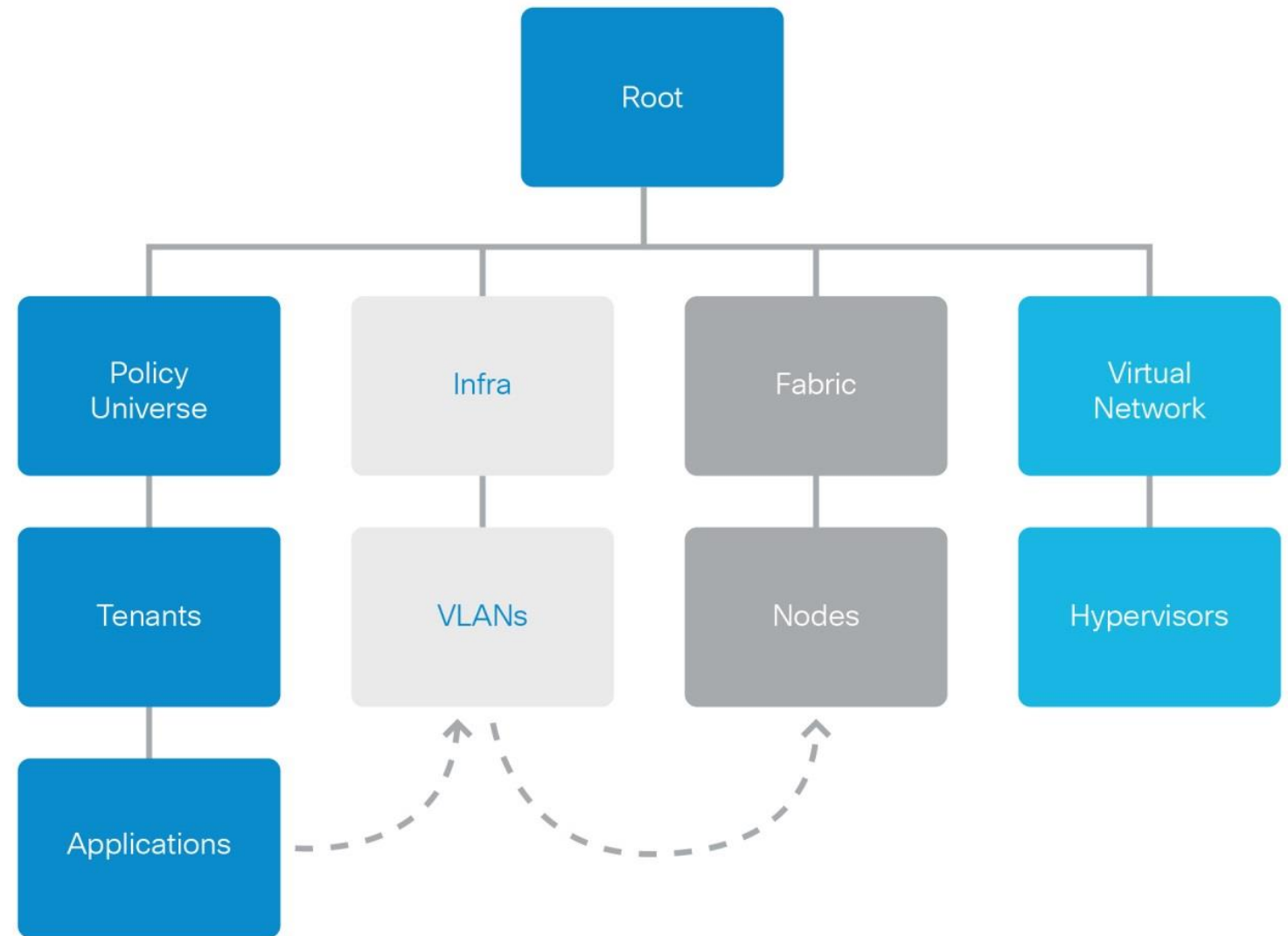
ACIプログラマビリティ オプション

オブジェクトモデル



オブジェクトモデル

- ◆ Rootが最上位



テナントツリーの一部

◆ オブジェクトを関連付けるさまざまな方法

◆ 関係

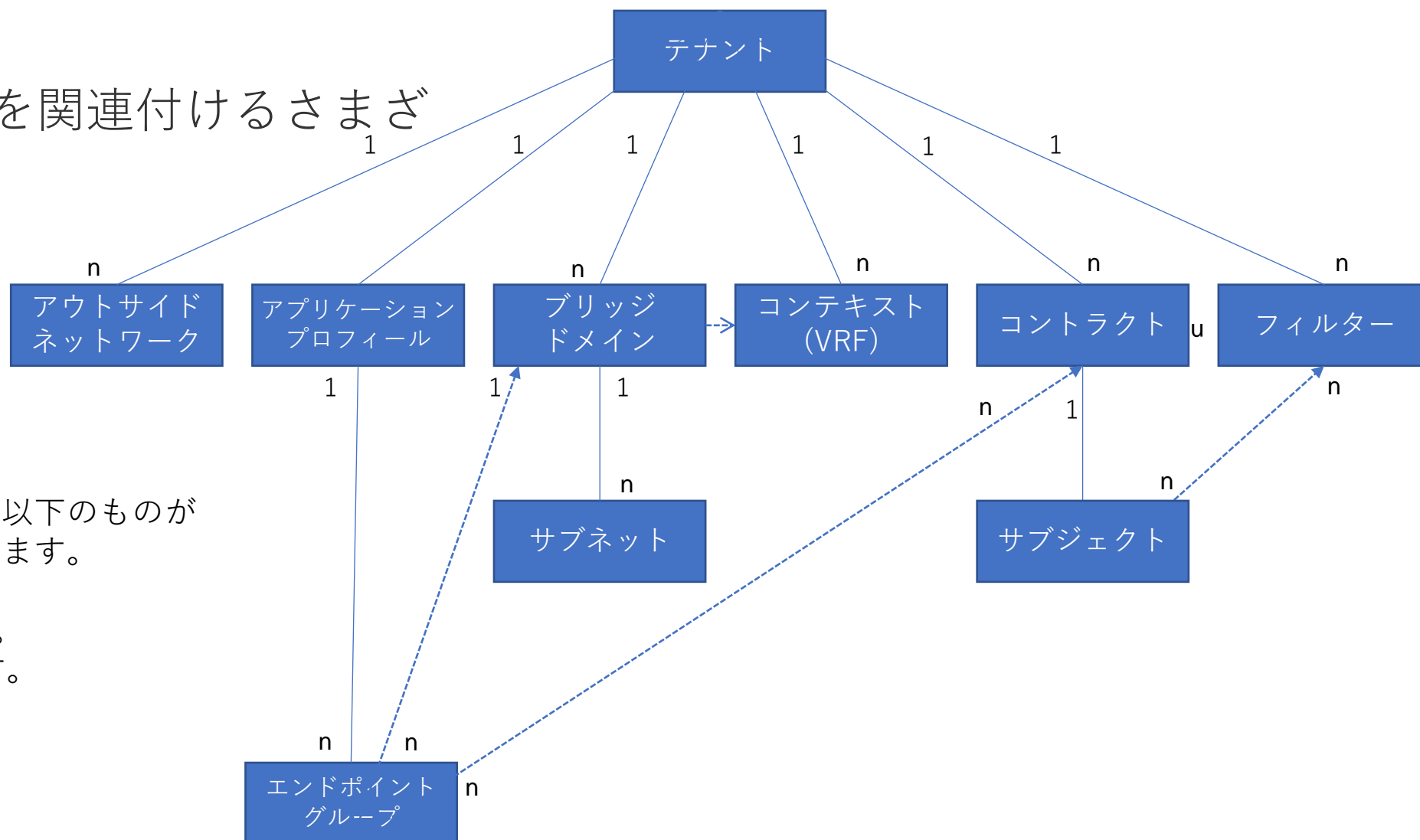
- 1対1
- 1対多
- 多対多

実線は、オブジェクトに以下のものが含まれていることを示します。

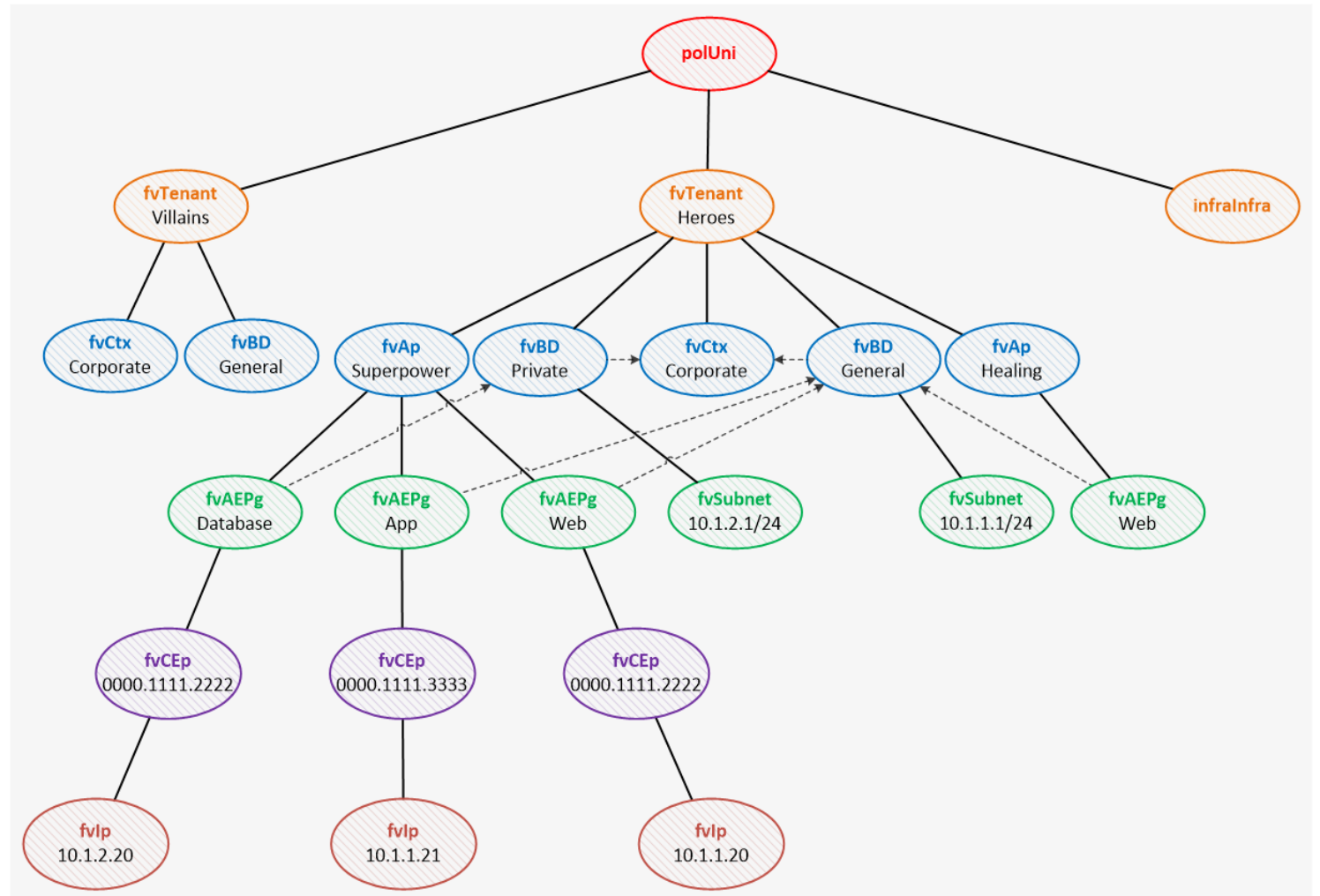
点線は関係を示します。

1 : nは1対多を示します。

n : nは多対多を示します。



MITオブジェクト



Cobra

Cobra

- ◆ ACI Cobra SDK
 - acicobra
 - acimodel
- ◆ 完全な Cisco ACI Python SDK
- ◆ API の Python の本来の実装
- ◆ REST のすべての機能固有のバインディングを提供
- ◆ オブジェクト モデルの完全なコピーを備えている
 - データの整合性が確保
 - ACI で使用可能な機能と関数をすべてサポート

Cobra (続き)

- ◆ Cobra はルックアップやクエリ、GUI が使用する REST に一致し、API インспекタを使用して検出可能なオブジェクトの作成、変更、および削除を実行するメソッドを提供します。その結果、GUI で作成したポリシーをプログラミング テンプレートとして使用して、開発を促進することができます。
- ◆ Cobra のインストール プロセスは簡単で、標準的な Python 配布ユーティリティを使用できます。Cobra は APIC 上に .egg ファイルとして配布され、easy_install を使用してインストールできます。また、<http://github.com/datacenter/cobra> の github でも入手できます。

Cobra -セッションの確立

- ◆ Cobra を使用するコードでは、まず、ログインセッションを確立します。現在、Cobra はユーザ名とパスワードベースの認証と、証明書ベースの認証をサポートしています。
- ◆ 次に、ユーザ名とパスワード名をベースにした認証を使用する例を示します。

```
import cobra.mit.access
import cobra.mit.session
URL = 'https://10.0.0.2'
LOGIN = 'username'
PASSWORD = 'password'
auth = cobra.mit.session.LoginSession(URL, LOGIN, PASSWORD)
session = cobra.mit.access.MoDirectory(auth)
session.login()
```

- ◆ この例では、Cisco APIC にログインし、認証される **session** という **MoDirectory** オブジェクトを実装します。何らかの理由で認証が失敗した場合、Cobra は `cobra.mit.request.CommitError` 例外メッセージを表示します。セッションにログインしたら、続行できます。

Cobra – オブジェクトの使用

- ◆ Cobra SDK を使用して、このワークフローに概ね従っている MIT を操作します。
 1. 操作するオブジェクトを特定します。
 2. 属性を変更する、あるいは子オブジェクトを追加または削除する要求を構築します。
 3. オブジェクトに加えた変更をコミットします。
- ◆ たとえば、新しいテナントを作成する場合は、まず、MIT のどこにテナントが配置されるか、この場合は、**policy Universe** の監視対象オブジェクトの子オブジェクト(polUniMo)を特定します。

```
import cobra.model.pol
polUniMo = cobra.model.pol.Uni("")
```
- ◆ **polUniMo** オブジェクトを定義すると、テナント オブジェクトを **polUniMo** の子として作成できます。

Cobra – テナントの作成

```
# Create a Variable for your Tenant Name
# Use your initials in the name
# Example: tenant_name = "js_Cobra_Tenant"
tenant_name = "自分の名前_Cobra_Tenant"

# create a new tenant
polUniMo = cobra.model.pol.Uni('')
new_tenant = cobra.model.fv.Tenant(polUniMo, tenant_name)

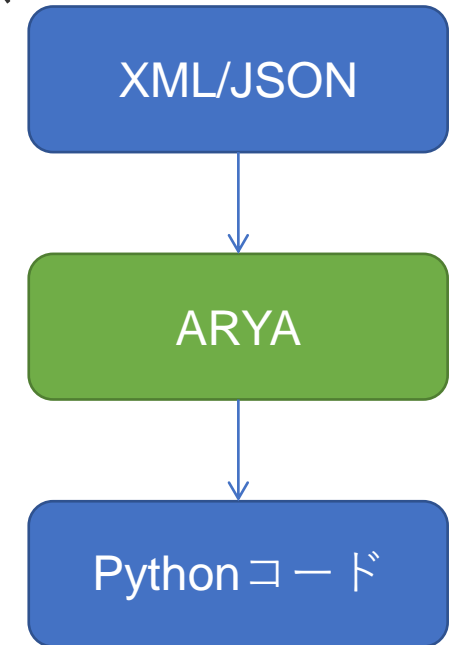
# commit the new configuration
config_request = cobra.mit.request.ConfigRequest()
config_request.addMo(new_tenant)
session.commit(config_request)

# <Response [200]> --> 200が返ってきて成功
```

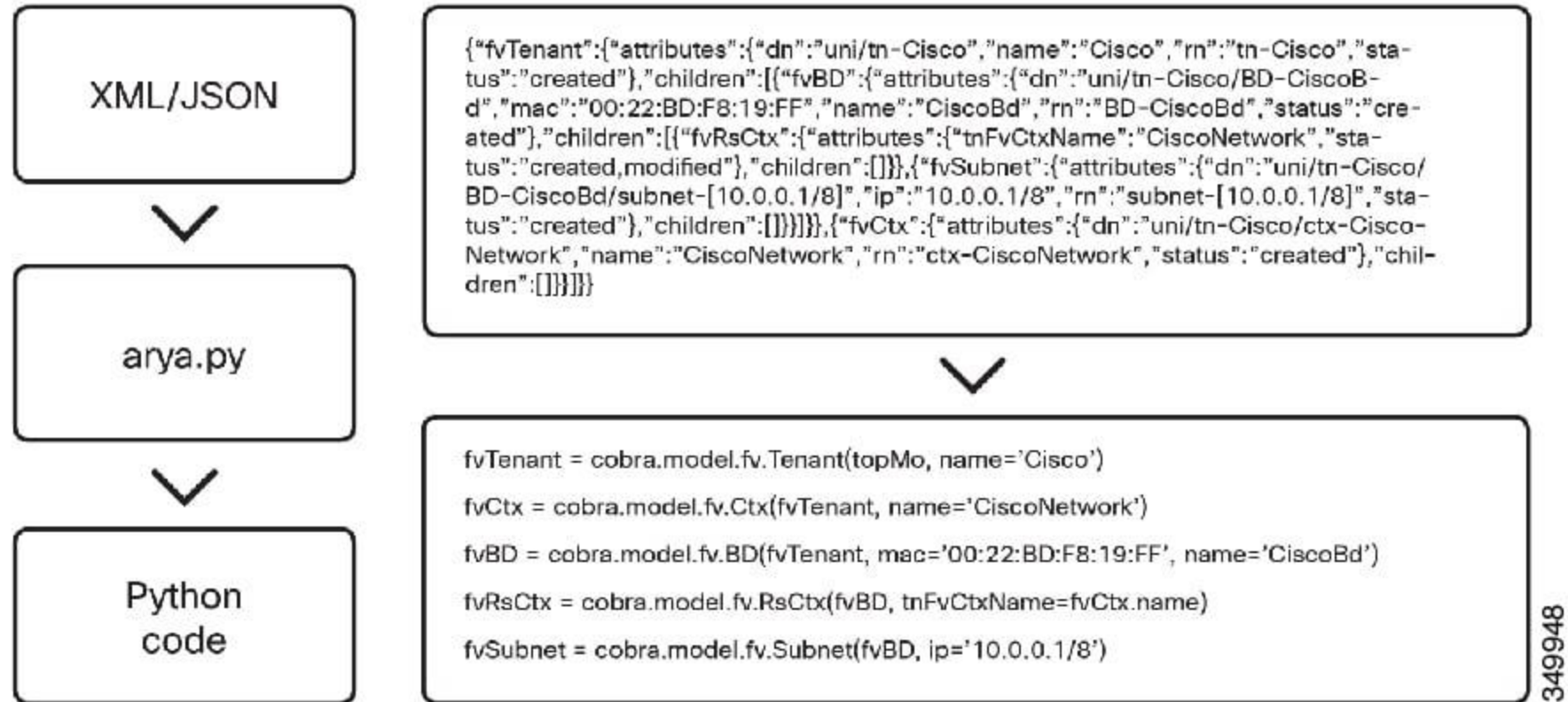
Arya

Aryaとは？

- ◆ (A)PIC (R)est p(Y)thon (A)dapter
 - Cobraライブラリを使用したPythonスクリプトの生成を支援
 - **XMLまたはJSONの構成データ**のいずれかを受け取り、同じ構成を生成するための**Pythonコード**を出力
- ◆ ARYAの利点
 - 構成スクリプトの作成にかかる時間を短縮
 - APIドキュメントを読むよりも簡単
 - APIの使用方法のサンプル



Python アダプタへのサンプル REST



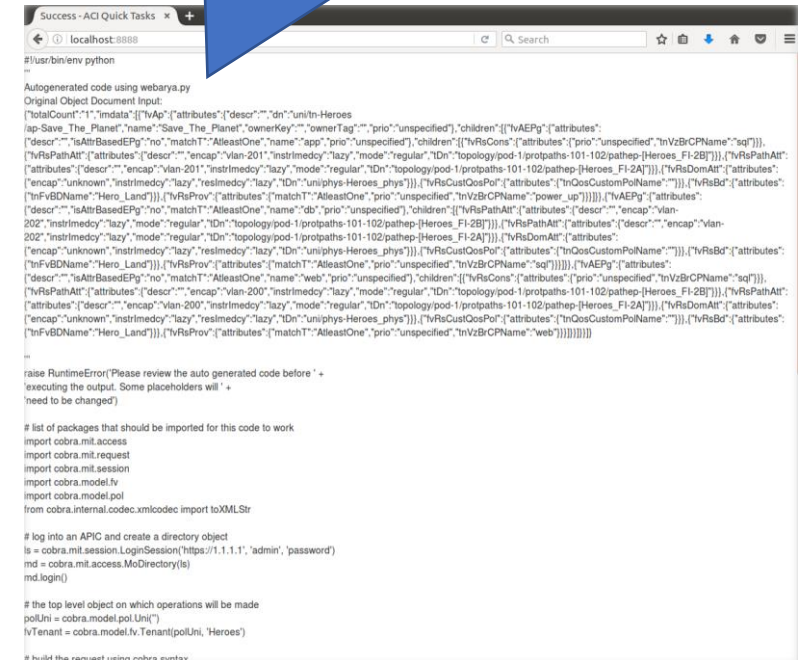
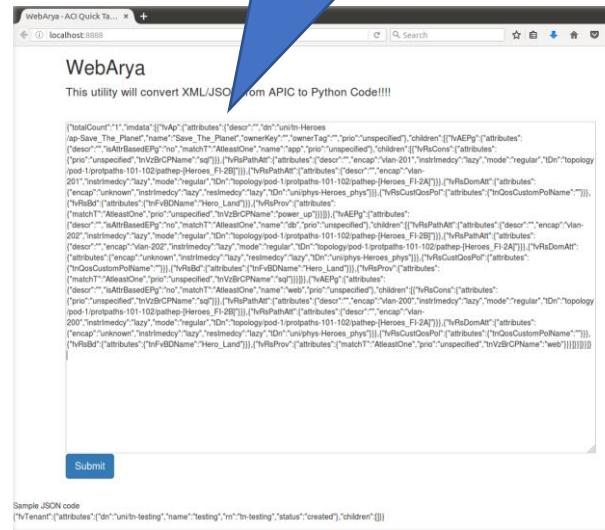
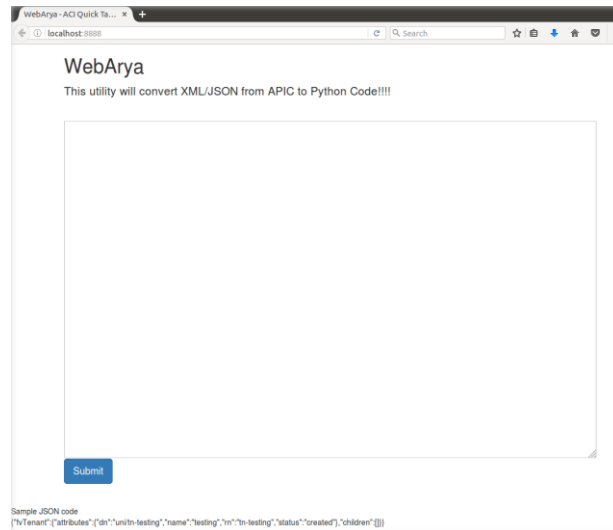
349948

WebArya

- ◆ Aryaと同じですが、使いやすいWebインターフェイスを提供

Pythonプログラムが得られる

貼り付けてSubmit



インストール(以下のいずれか)

◆ git

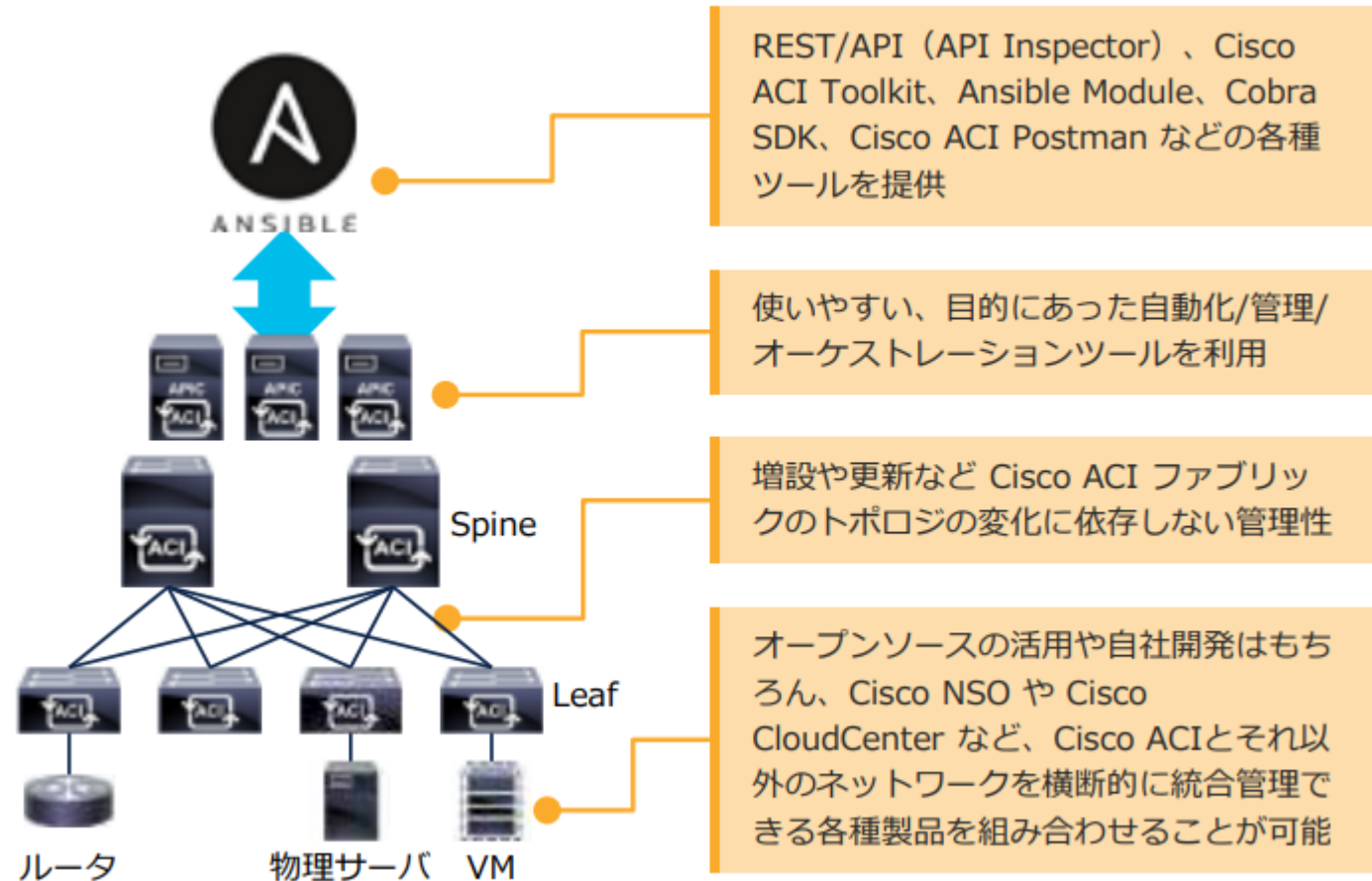
- `git clone https://github.com/datacenter/arya.git`
- `cd arya`
- `python setup.py install`
- `arya.py` (チェック)

◆ pip install arya

ACI と Ansible

Ansibleによる構成管理

- ◆ Ansible 2.4 以降では、Cisco APIC および MSO (Multi-Site Orchestrator) に対応する 連携モジュールが標準で含まれており、Cisco ACIの構成管理にすぐ活用できます。大半の 構成が可能な100以上のモジュールが提供されており、モジュールが用意されていない構成でも、Cisco APICのREST/APIにアクセスする「aci_rest」モジュールを活用することで汎用的に対応可能です。

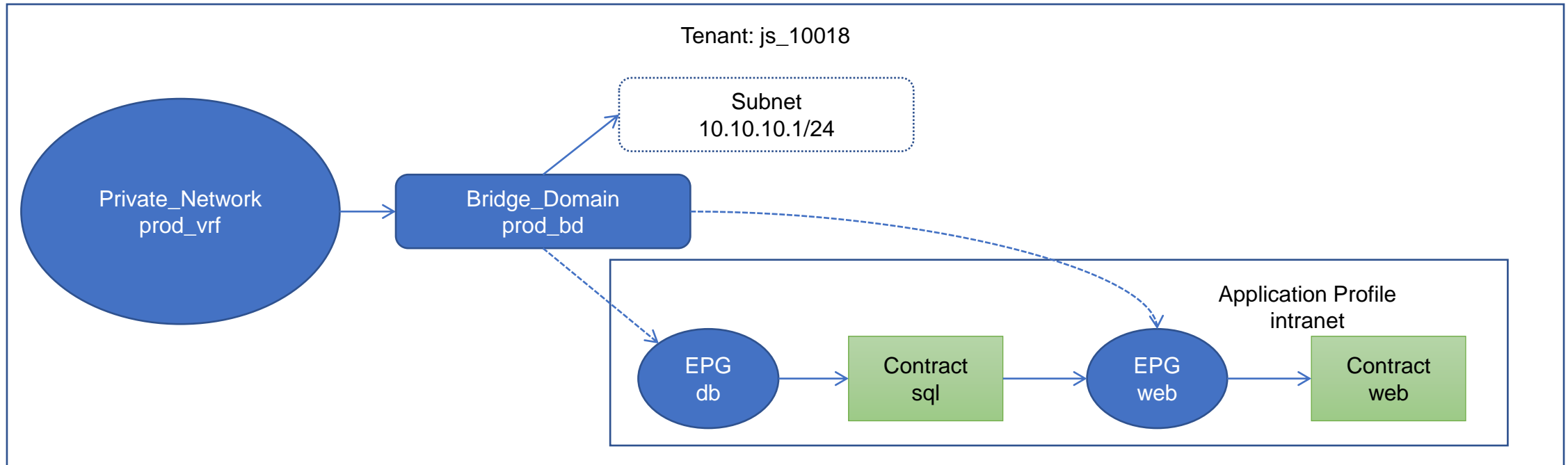


ACIモジュールの四つのタイプ

テナントモジュール	アプリプロファイル、EPG、VRF、ブリッジドメイン、ポリシー構成などのテナント構成を管理するために使用
インフラストラクチャモジュール	ファブリックインベントリおよびスイッチポートポリシーとの対話に使用
構成管理	パッケージは現在、構成スナップショットの作成、構成の違いのプレビュー、および前のスナップショットへのロールバックの実行をサポート
汎用モジュール	まだモジュールがない機能に使用されます。JSONまたはXMLファイルに保存されたデータを使用してAPIにリクエストを送信するために使用できる汎用モジュール

一般的なACIモジュールパラメータ

hostname	ホストの名前またはIPアドレス
username	APICへのログインに使用するユーザ名
password	ユーザのアカウントに関連付けられているパスワード
use_ssl	httpとhttpsのどちらを使用するかを決定します。デフォルトはhttpsですが、httpを使用するにはFalseに設定できます
validate_certs	<p>サーバのknown_hostsファイルに対してAPICの証明書を検証するかどうかを決定します。デフォルトはTrue検証ですが、False無効な証明書の警告を無視するように設定できます</p> <p>present：オブジェクトの構成が存在することを確認</p> <p>absent：オブジェクトの構成が存在しないことを確認</p> <p>query：オブジェクトまたはオブジェクトクラスの既存の構成のリストを取得</p>
description	オブジェクトの構成に適用する説明。オブジェクトの目的を識別するために使用されます。



inventory ファイル

```
[apic:vars]
```

```
username=admin
```

```
password=!v3G@!4@Y
```

```
ansible_python_interpreter="/usr/bin/env python"
```

```
[apic]
```

```
sandboxapicdc.cisco.com
```

[apic:vars]は、apicと呼ばれるグループのグループ変数を示すことに注意してください。
このグループapicには、パブリックに解決可能なDNS名であるsandboxapicdc.cisco.comと
いう単一のホストがあります。



Playbook (アプリケーションのAPICに必要な構成が存在することを確認するために使用)

- name: ENSURE APPLICATION CONFIGURATION EXISTS

hosts: apic

connection: local

gather_facts: False

vars_prompt:

- name: "tenant"

prompt: "What would you like to name your Tenant?"

private: no

tasks:

- name: ENSURE APPLICATIONS TENANT EXISTS

aci_tenant:

host: "{{ inventory_hostname }}"

username: "{{ username }}"

password: "{{ password }}"

state: "present"

validate_certs: False

tenant: "{{ tenant }}"

description: "Tenant Created Using Ansible"



Ansible ACI Playbookの実行

```
$ ansible-playbook -i inventory 01_aci_tenant_pb.yml
```

```
What would you like to name your Tenant?: jt_1234
```

```
PLAY [ENSURE APPLICATION CONFIGURATION EXISTS] *****
```

```
TASK [ENSURE APPLICATIONS TENANT EXISTS] *****
```

```
changed: [apic1.dcloud.cisco.com]
```

```
PLAY RECAP *****
```

```
1pic1.dclud.cisco.com      : ok=1    changed=1    unreachable=0    failed=0    skipped=0    rescued=0  
ignored=0
```

```
$
```

02_aci_tenant_network_pb.yml

```
---
- name: ENSURE APPLICATION CONFIGURATION EXISTS
  hosts: apic
  connection: local
  gather_facts: False

  vars_prompt:
    - name: "tenant"
      prompt: "What would you like to name your Tenant?"
      private: no

  tasks:
    - name: ENSURE TENANT VRF EXISTS
      aci_vrf:
        host: "{{ inventory_hostname }}"
        username: "{{ username }}"
        password: "{{ password }}"
        state: "present"
        validate_certs: False
        tenant: "{{ tenant }}"
        vrf: "{{ vrf }}"
        description: "VRF Created Using Ansible"
```



02_aci_tenant_network_pb.yml (続き)

```
- name: ENSURE TENANT BRIDGE DOMAIN EXISTS
  aci_bd:
    host: "{{ inventory_hostname }}"
    username: "{{ username }}"
    password: "{{ password }}"
    validate_certs: False
    state: "present"
    tenant: "{{ tenant }}"
    bd: "{{ bd | default('prod_bd') }}"
    vrf: "{{ vrf }}"
    description: "BD Created Using Ansible"
```

```
- name: ENSURE TENANT SUBNET EXISTS
  aci_bd_subnet:
    host: "{{ inventory_hostname }}"
    username: "{{ username }}"
    password: "{{ password }}"
    validate_certs: False
    state: "present"
    tenant: "{{ tenant }}"
    bd: "{{ bd | default('prod_bd') }}"
    gateway: "10.1.100.1"
    mask: 24
    scope: "public"
    description: "Subnet Created Using Ansible"
```

ACI Ansibleテナントネットワークモジュールを実行

```
$ ansible-playbook 02_aci_tenant_network_pb.yml -i inventory --extra-vars "vrf=prod_vrf"
```

```
What would you like to name your Tenant?: jt_1234
```

```
PLAY [ENSURE APPLICATION CONFIGURATION EXISTS] *****
```

```
TASK [ENSURE TENANT VRF EXISTS] *****
```

```
changed: [apic1.dcloud.cisco.com]
```

```
TASK [ENSURE TENANT BRIDGE DOMAIN EXISTS] *****
```

```
changed: [apic1.dcloud.cisco.com]
```

```
TASK [ENSURE BRIDGE DOMAIN SUBNET EXISTS] *****
```

```
changed: [apic1.dcloud.cisco.com]
```

```
PLAY RECAP *****
```

```
apic1.dcloud.cisco.com : ok=3  changed=3  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
```



ACI Ansibleテナントポリシー/ Contractモジュールを確認

```
- name: ENSURE TENANT FILTERS EXIST
  aci_filter:
    host: "{{ inventory_hostname }}"
    username: "{{ username }}"
    password: "{{ password }}"
    action: "post"
    protocol: "https"
    tenant: "{{ tenant }}"
    filter: "{{ item }}"
    descr: "Filter Created Using Ansible"
  with_items:
    - "https"
    - "sql"
```

```
- name: ENSURE FILTERS HAVE FILTER ENTRIES
  aci_filter_entry:
    host: "{{ inventory_hostname }}"
    username: "{{ username }}"
    password: "{{ password }}"
    state: "present"
    validate_certs: False
    tenant: "{{ tenant }}"
    filter: "{{ item.filter }}"
    entry: "{{ item.entry }}"
    ether_type: "ip"
    ip_protocol: "tcp"
    dst_port_start: "{{ item.port }}"
    dst_port_end: "{{ item.port }}"
  with_items:
    - filter: "https"
      entry: "https"
      port: 443
    - filter: "sql"
      entry_name: "sql"
      port: 1433
```



ACI Ansibleテナントポリシー/コントラクトモジュールを実行

```
$ ansible-playbook -i inventory 03_aci_tenant_policies_pb.yml -vvv
```

```
PLAYBOOK: 03_aci_tenant_policies_pb.yml *****
```

```
1 plays in 03_aci_tenant_policies_pb.yml
```

```
What would you like to name your Tenant?: jt_1234
```

```
PLAY [ENSURE APPLICATION CONFIGURATION EXISTS] *****
```

```
META: ran handlers
```

```
TASK [ENSURE TENANT FILTERS EXIST] *****
```

```
changed: [apic1.dcloud.cisco.com] => (item=https) => {
```

```
  "changed": true,
```

```
  ...
```

(出力省略)

```
  ...
```

```
PLAY RECAP *****
```

```
apic1.dcloud.cisco.com : ok=5 changed=5 unreachable=0 failed=0
```



参考URL

- ◆ Cisco ACI モジュールの開発
 - https://docs.ansible.com/ansible/2.9_ja/dev_guide/developing_modules_general_aci.html

Terraform

Terraform

- ◆ HashiCorpが作成したオープンソースソフトウェア
- ◆ ソフトウェアとしてのインフラストラクチャ（IaC）であり、データセンターまたはクラウド環境を管理
- ◆ 構成ファイルを使用して、インフラストラクチャの望ましい状態を記述
 - 構成ファイルを受け取り、目的の状態に到達するために何をする必要があるかを記述した実行プランを生成
 - 計画が確認された後、必要なアクションを実行して目的の状態に到達
 - また、何が変更されたかを判断し、増分実行プランを作成可能
 - 作成したインフラストラクチャを削除も可能
- ◆ 構成ファイルをバージョン管理下に置くことができるため、インフラストラクチャの「以前のバージョン」にすばやく移行可能



Terraformとその他の比較

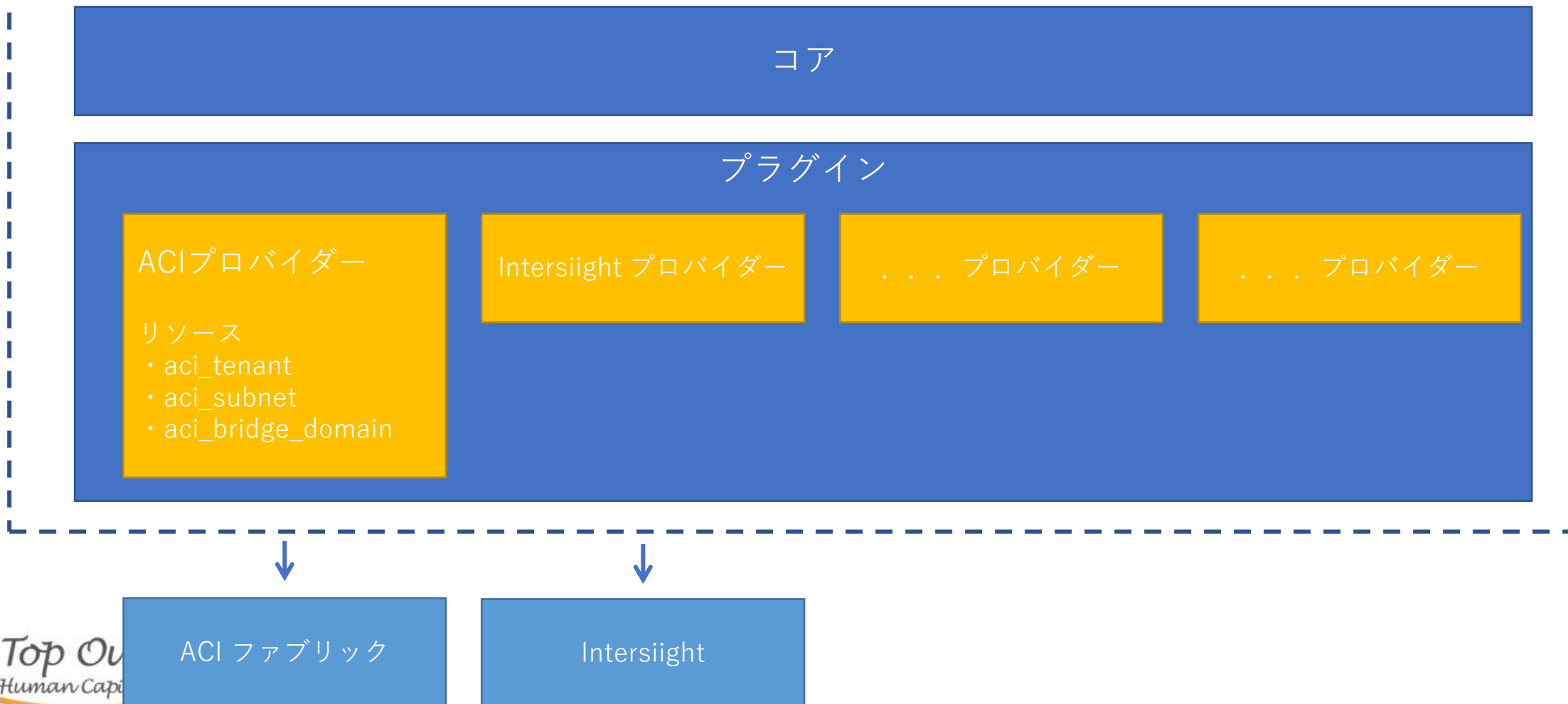
	Terraform	ansible	CloudFormation
ACI	○	○	
AWS	○	○	○
Azure	○	○	
GCP	○	○	
GitHub	○	○	○
Datadog	○	○	
esxi	○	○	
Hyper-V	○	○	

Infrastructure as Code (IaC)

- ◆ **導入時間が短縮される。**
 - 意図した設定をファイルに記録し、IaCツールを使用することで、GUIをクリックしていく必要がなくなるため、導入にかかる時間が短縮されます。
 - 重大なエラーが発生した場合に、インフラを迅速に再導入して設定できます。
- ◆ **設定とのずれがなくなる。**
 - IaC ツールはインフラ(VCenterなど)をモニタし、ファイル内の意図した設定がインフラと一致していることを確認できます
- ◆ **チームでのコラボレーションが可能になる。**
 - すべての設定がテキストファイルに定義されているため、チームのメンバーがインフラの設定方法をすばやく理解できます。
- ◆ **説明責任と変更内容が明確になる。**
 - 設定を記載したテキストファイルは、Git などのバージョン管理ソフトウェアを使用して保存できます。また、二つのバージョン間の設定の違いも確認できます。
- ◆ **複数の製品を管理できる。**
 - すべてではないにしても、ほとんどの IaC ツールは、複数の製品やドメインに対応しているため、上記のメリットを一つのツールで一元的に得られます

リソースとプロバイダ

Terraform



Terraformプログラムの例 (main.tf)

```
# definition provider
provider "aws" {
  version = "~> 2.0"
  region = "${var.provider_region}"
  access_key = "${var.secret_access_key}"
  secret_key = "${var.secret_key}"
}
```

UbuntuとPostgresデータベースを宣言 (resource.tf)

リソースタイプ

リソース名

```
resource "aws_instance" "web" {
  ami = "${data.aws_ami.ubuntu.id}"
  instance_type = "${var.server_instance_type}"

  tags = {
    Name = "${var.server_tag_name}"
  }
}

data "aws_ami" "ubuntu" {
  most_recent = true

  filter {
    name = "name"
    values = ["ubuntu/images/hvm-ssd/ubuntu-trusty-14.04-amd64-server-*"]
  }

  owners = ["099720109477"]
}
```



UbuntuとPostgresデータベースを宣言 (resource.tf) (続き)

```
resource "aws_db_instance" "default" {  
    allocated_storage = "${var.db_allocated_storage}"  
    storage_type = "${var.db_storage_type}"  
    engine = "${var.db_engine}"  
    engine_version = "${var.db_engine_version}"  
    instance_class = "${var.db_instance_class}"  
    name = "${var.db_database_name}"  
    username = "${var.db_database_username}"  
    password = "${var.secret_db_database_password}"  
    final_snapshot_identifier = "${var.db_snapshot_identifier}"  
}
```

変数の入力例 (variables.tf)

```
# provider variables
variable "provider_region" {
  description = "Provider region"
  default = "us-east-1"
}
```

```
variable "secret_access_key" {
  description = "Provider access key"
  default = "YOUR-ACCESS-KEY"
}
```

```
variable "secret_key" {
  description = "Provider secret key"
  default = "YOUR-SECRET-KEY"
}
```

```
#instance web variables
variable "server_instance_type" {
  description = "Server instance type"
  default = "t2.micro"
}
```

```
variable "server_tag_name" {
  description = "Server tag name"
  default = "JeSuisUnDev"
}
```

```
# instance base de données RDS
postgres variables
variable "db_allocated_storage" {
  description = "Allocated storage"
  default = 20
}
```

変数の入力の例 (variables.tf) (続き)

```
variable "db_storage_type" {  
  description = "Storage type"  
  default = "gp2"  
}
```

```
variable "db_engine" {  
  description = "Storage engine"  
  default = "postgres"  
}
```

```
variable "db_engine_version" {  
  description = "Storage engine version"  
  default = "11.5"  
}
```

```
variable "db_instance_class" {  
  description = "Storage instance class"  
  default = "db.t2.micro"  
}
```

```
variable "db_database_name" {  
  description = "Storage database name"  
  default = "postgres"  
}
```

```
variable "db_database_username" {  
  description = "Storage database name"  
  default = "postgres"  
}
```

```
variable "secret_db_database_password" {  
  description = "Storage database secret password"  
  default = "postgres"  
}
```

```
variable "db_snapshot_identifier" {  
  description = "Storage database snapshot identifier"  
  default = "postgres"  
}
```



四つの主要なTerraformコマンド

- ◆ **init** - プランを実行できるようにTerraformディレクトリを初期化します。Terraformは、main.tfで定義されているすべてのプロバイダをダウンロードしようとします。
- ◆ **plan** - main.tfファイルを分析し、状態ファイルterraform.tfstate(存在する場合)と比較して、計画のどの部分を展開、更新、または破棄する必要があるかを判断します。
- ◆ **apply** - planコマンドで記述された変更をサードパーティシステムに適用terraform.tfstateし、プランで記述されたリソースの現在の構成状態でファイルを更新します。
- ◆ **destroy** - 以前にデプロイされたすべてのリソースを削除またはunconfigure[構成解除]します。Terraformは、状態ファイルterraform.tfstateを使用してこれらのリソースを追跡します。

Terraform 初期化 (init)

> **terraform init**

Initializing the backend...

Initializing provider plugins...

- Checking for available provider plugins...
- Downloading plugin for provider "aws" (hashicorp/aws) 2.48.0...

Terraform has been successfully initialized!

Terraform 検証 (validate)

> terraform validate

Success! The configuration is valid.

Terraform Plan (実行する前に確認を行う)

> terraform plan

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:

+ create

Terraform will perform the following actions:

aws_db_instance.default will be created

aws_instance.web will be created

Plan: 2 to add, 0 to change, 0 to destroy.



Terraform Apply (planを実行)

> terraform apply

```
aws_instance.web: Creating...
aws_db_instance.default: Creating...
aws_instance.web: Still creating... [10s elapsed]
aws_instance.web: Still creating... [30s elapsed]
aws_instance.web: Creation complete after 43s [id=i-0f285230749e69a67]
aws_db_instance.default: Still creating... [50s elapsed]
aws_db_instance.default: Still creating... [1m50s elapsed]
aws_db_instance.default: Still creating... [2m50s elapsed]
aws_db_instance.default: Still creating... [3m50s elapsed]
aws_db_instance.default: Still creating... [4m0s elapsed]
aws_db_instance.default: Creation complete after 4m8s [id=terraform-
202002080646247297000000001]
```

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Terraform show

```
$ terraform show terraform.tfstate
```

```
aws_instance.example:  
  id = i-02bbfece7cbf56027  
  ami = ami-408c7f28  
  instance_type = t1.micro  
  private_ip = 10.178.172.29  
  public_ip = 10.17.172.177
```

~

リソースの削除 (terraform destroy)

> terraform destroy

An execution plan has been generated and is shown below.

Resource actions are indicated with the following symbols:

- destroy

aws_instance.web: Destroying... [id=i-0f285230749e69a67]

aws_db_instance.default: Destroying... [id=terraform-202002080646247297000000001]

aws_instance.web: Destruction complete after 30s

aws_db_instance.default: Destruction complete after 50s

Destroy complete! Resources: 0 added, 0 changed, 2 destroyed.

