

# Cisco DNA Programmability Training

## -ACI編-

ラボガイド

2021/08/30| Document# | Version 1.1

Top Out Human Capital株式会社



## 更新履歴

Version 番号	日付	内容
1.0	2021/5/10	第一版
1.1	2021/8/30	dCloud 対応

## 目次

1. 事前準備 .....	1-1
ステップ1: dcldcloudへの接続 .....	1-1
ステップ2: アプリケーションのインストール .....	1-3
ステップ2: ファイルの確認 .....	1-4
ステップ3: テナントやアプリケーション名の指定(演習の中で利用します) .....	1-4
ステップ4: ファイルのコピー .....	1-4
ステップ5: venv: Python仮想環境 .....	1-4
2. ACIプログラマビリティ .....	2-5
2.1. ACIの理解 .....	2-6
ステップ1: はじめに .....	2-6
ステップ2 : ACIをさらに深く掘り下げる .....	2-6
ステップ3 : ACIの用語 .....	2-7
ステップ4 : テナントネットワーキング .....	2-8
ステップ5 : テナントポリシー .....	2-9
演習1 : ACI GUIウォークスルー .....	2-11
2.2. ACIプログラマビリティオプション .....	2-18
演習0: APICサンドボックス環境のセットアップ .....	2-18
ステップ1 : はじめに .....	2-18
ステップ2 : 管理対象オブジェクト .....	2-19
ステップ3 : 管理情報ツリー .....	2-20
ステップ4 : 相対名 .....	2-23
演習1 : ACI REST API .....	2-24
演習2 : ACIツールキットを使用する .....	2-26
演習3 : Cobra SDKを使用する .....	2-27
演習4 : 作成したテナントをGUIで表示する .....	2-28
まとめ:ACIプログラマビリティエイド .....	2-30
3. ACIプログラム応用 .....	3-31
3.1. 簡単な方法でACIスクリプトを作成する-WebArya .....	3-32
ステップ1: Aryaとは何ですか? .....	3-33
演習1 : WebAryaをインストール .....	3-33
演習2 : GUIでJSONをダウンロードする .....	3-34
演習3 : Cobraコードを収集する .....	3-36
演習4 : ターミナルとコードノート .....	3-40
演習5:ターミナルとコードノート .....	3-44
3.2. Cobraを使用してACI Pythonスクリプトに「Bite」を追加 .....	3-46
ステップ1: Cobra入門 .....	3-46
ステップ2: 「acicobra」と「acimodel」の区別 .....	3-46
ステップ3: ACI Cobraパッケージの使用 .....	3-47
演習1 セッションの確立 .....	3-47
演習2 クエリを作成する .....	3-48
演習3 スコープクエリ .....	3-48
3.3. 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します。 .....	3-50
ステップ1: ミッションの概要 .....	3-50
ステップ2: ダッシュボードアプリケーション .....	3-50
ステップ3 解決 .....	3-52
ダッシュボードを表示する .....	3-53
4. ACIとAnsibleの紹介 .....	4-55
4.1. AnsibleでCisco ACIの自動化を実行 .....	4-56
ステップ1 : AnsibleおよびACI環境をセットアップする .....	4-56
ステップ2: Ansible ACI Playbookを確認する .....	4-56
演習1: Ansible ACI Playbookを表示する .....	4-56
演習2: Ansible ACI Playbookを実行する .....	4-58
4.2. AnsibleでApplicationとNetwork Policy .....	4-60
ステップ1 : ACIテナントネットワークモジュールを確認する .....	4-60
ステップ2 : ACI Ansibleネットワークモジュールを確認する .....	4-60
ステップ3 : ACI Ansibleテナントネットワークモジュールを実行する .....	4-62
ステップ4 : ACI Ansibleテナントポリシー/Contractモジュールを確認する .....	4-63
ステップ5 : ACI Ansibleテナントポリシー/コントラクトモジュールを実行する .....	4-65
ステップ6 : ACI Ansibleテナントアプリケーションプロファイルモジュールを確認する .....	4-67
ステップ7 : ACI Ansibleテナントアプリケーションポリシーモジュールを実行する .....	4-69

4.3. AnsibleでACI as Codeとしてまとめる .....	4-71
ステップ1：アプリケーションポリシー用の単一のPlaybookを作成する .....	4-71
ステップ2：Ansibleタグと変数を確認する .....	4-71
ステップ3：タグを使用してPlaybookを実行する .....	4-72
ステップ4：ACI REST APIコマンドを実行する .....	4-73
ステップ5：ラボのクリーンアップを実行する .....	4-74
5. ACIとTerraformの紹介 .....	5-76
5.1. TerraformでCisco ACIの自動化を実行 .....	5-77
ステップ1: ACI Terraformプロバイダの概要 .....	5-77
ステップ2: TerraformプランでACIリソースを確認する .....	5-79
ステップ3: Terraformプランコマンドを実行します .....	5-80
ステップ4 : ACI Terraformプランを適用し、テナントを作成する .....	5-81
5.2. TerraformでApplicationとNetwork Policy .....	5-83
ステップ1: ACITerraformテナントネットワークリソースを確認する .....	5-83
ステップ2: プランを適用し、テナントネットワークポリシーを作成します .....	5-84
ステップ3: ACITerraformテナントポリシー/コントラクトリソースを確認する .....	5-87
ステップ4: プランを適用し、コントラクト/フィルター/ポリシーリソースを作成します .....	5-89
ステップ5: ACI Terraformアプリケーションプロファイルリソースを確認する .....	5-90
ステップ6: プランを適用し、アプリケーションポリシーを作成します .....	5-93
5.3. ACI REST API を汎用ACI Terraformリソースをラップする .....	5-95
ステップ1: 一般的なACI Terraformリソースを確認する .....	5-95
ステップ2: 計画を適用し、一般的なACI Terraformリソースを実行します .....	5-96
ステップ3: 計画を破棄してラボをクリーンアップします .....	5-97



# 1. 事前準備

演習を始めるに当たり、必要となる事前準備を行います。

## ステップ 1: dcloud への接続

以下のURLに接続し、演習環境を利用します。

事前準備	
URL	<a href="https://dcloud2-sng.cisco.com/content/demo/231522">https://dcloud2-sng.cisco.com/content/demo/231522</a>
CCOアカウント	(各自のCCOアカウントをご利用ください)
パスワード	(各自のCCOアカウントをご利用ください)

The screenshot shows the Cisco DevNet Express Data Center v2 catalog page. At the top, there is a navigation bar with links for dCloud, My Hub, Catalog (which is highlighted with a red box), Support, News, Community, and Collections. On the right side of the header, there are icons for notifications, SNG, and user profile. Below the header, there is a breadcrumb trail with a 'Back' link and a 'Favorite' star icon. The main title is 'Cisco DevNet Express Data Center v2'. To the right of the title is a green 'Schedule' button, which is also highlighted with a red box. Under the title, there are two tabs: 'Information' and 'Resources', with 'Information' being the active tab. The 'Information' tab contains an 'Overview' section with a brief description of the programmatic capabilities of Cisco Data Center Infrastructure.

図1-1.

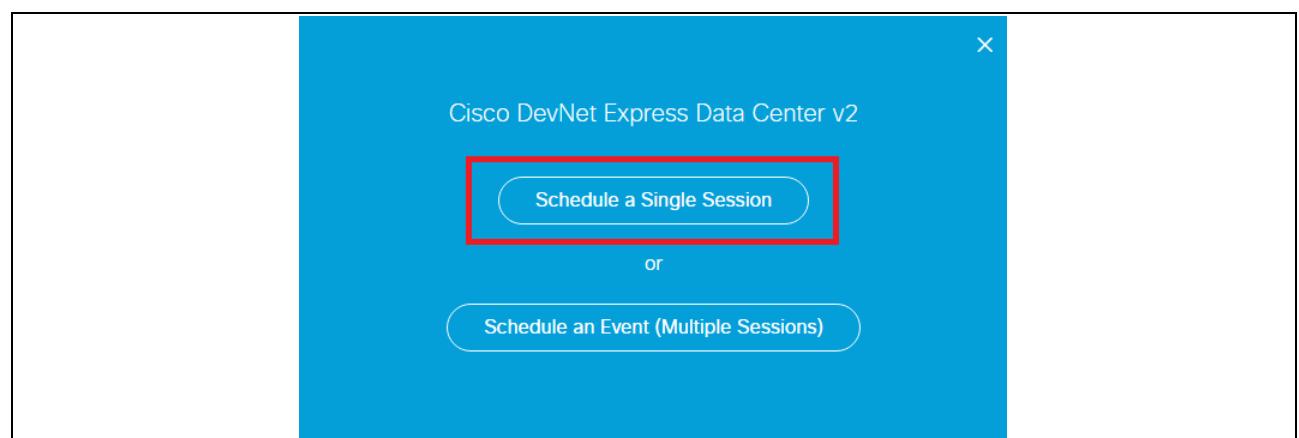


図1-2.

Back

## Cisco DevNet Express Data Center v2

Start Date [REDACTED] Start Time 10:00 Now

End Date [REDACTED] End Time 17:00

[Cancel](#) [Next](#)

図1-3.

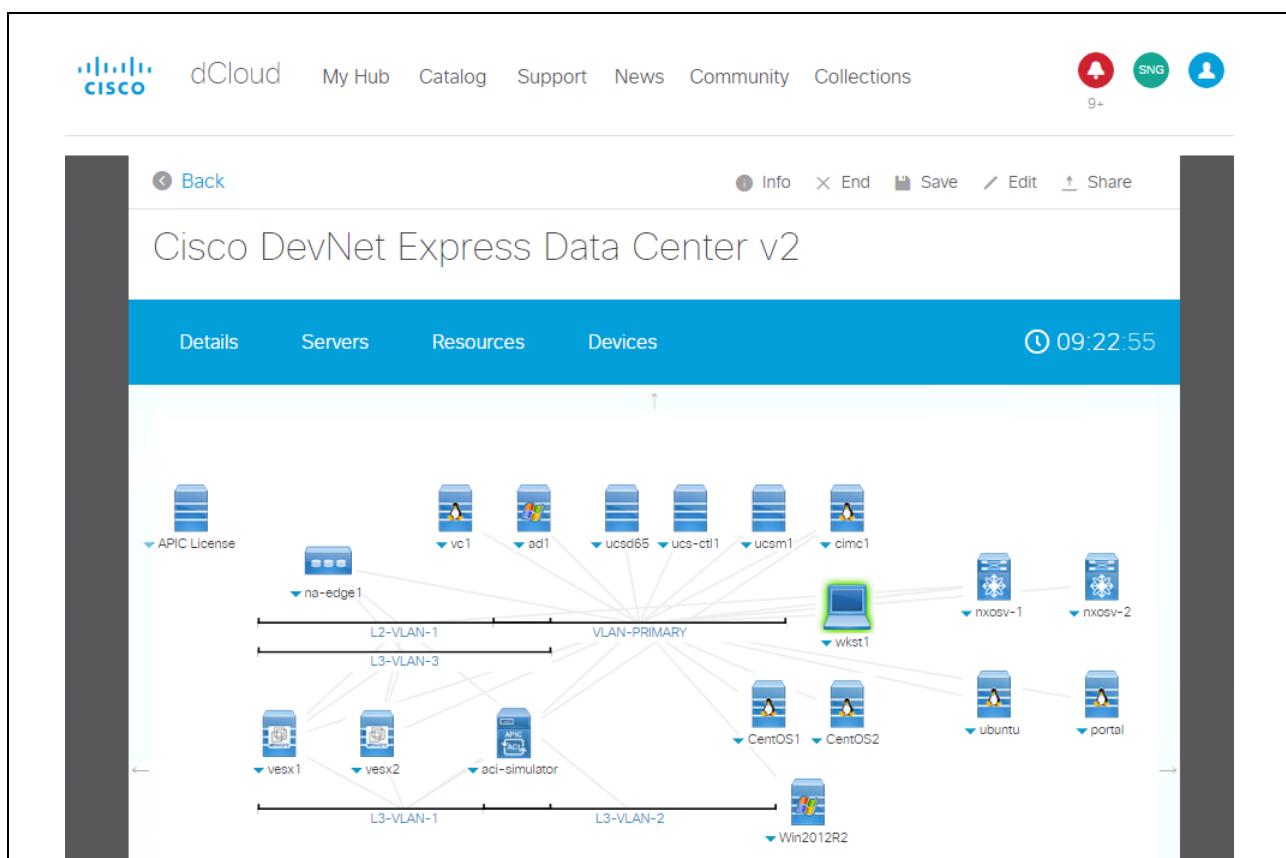


図1-4.



図1-5.

## ステップ 2: アプリケーションのインストール

今回利用するdccloud環境では、いくつかのファイルのインストールが必要です。

### Linuxマシン

Windowsマシンからputtyを利用して、以下のCentosに接続します。

Linux PCのクレデンシャル	
IPアドレス	198.18.134.49
ユーザ名	root
パスワード	C1sco12345

```
git clone https://github.com/TopOutHC/DevNetACI.git
```

```
cd DevNetACI
```

```
chmod +x AClinstall.sh
```

```
./AClinstall.sh
```

待っている間、Windowsマシンへの追加インストールを行います。

### Windowsマシン

#### Pythonの再インストール

インストール済みのPython2.7.13と、Python3.6.1をアンインストールします。

以下のURLからダウンロードし、Windows版64-bitのPython 3.8.10のインストールを行います。

<https://www.python.org/downloads/windows>

インストールを行う際に、「**Add Python 3.8 to PATH**」のチェックボックスにチェックを入れることを忘れないでください。

Windows PowerShellを起動し、以下のコマンドを入力します。

```
git clone https://github.com/TopOutHC/DevNetACI.git
```

```
cd DevNetACI
```

```
./AClinstall.ps1
```

#### Postmanの再インストール

以下のURLからダウンロードし、Windows版64-bitのアプリの再インストールを行う。

<https://www.postman.com/downloads>

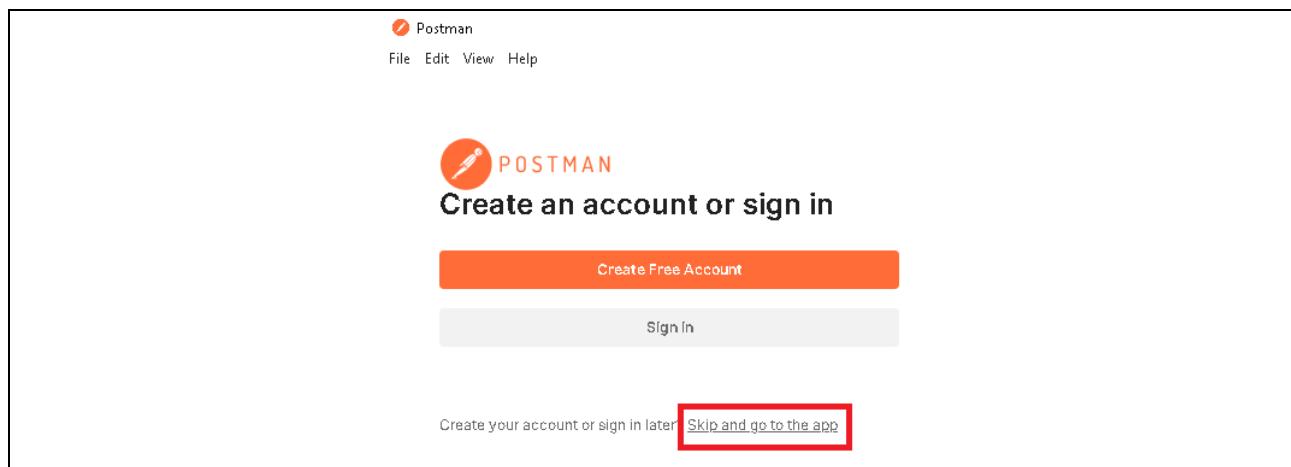


図1-6.

## ステップ 2: ファイルの確認

Windowsマシンに以下のファイルが存在していることを確認してください。演習で使用します。

- ・PDFファイル
- ・プログラムファイル

## ステップ 3: テナントやアプリケーション名の指定(演習の中で利用します)

全てのユーザが同一の環境を使用します。自分自身を区別するために、テナントやアプリケーション名にはイニシャルと電話番号を使用します。

テナントやアプリケーションの名前の入力を求められたら、イニシャルと電話番号(下4桁)を使用して名前を作成します。たとえば、電話番号1234のトップアウト次郎さんは、以下に示すように、jt\_1234を作成します。

例: tenant = "jt\_1234"

## ステップ 4: ファイルのコピー

セキュリティの理由から、リモートラボとは、直接ファイルのコピーを行うことはできません。以下の手順を行うことでファイルの受け渡しが可能です。

Windowsマシンでブラウザを開きます。以下のURLにアクセスし、Webex Teamsにブラウザ経由で参加します。

<https://teams.webex.com/signin>

## ステップ 5: venv: Python 仮想環境

演習の中で仮想環境を利用する場合があります。以下の様な手順で利用可能です。

### WindowsPCの場合

```
$ python -m venv venv  
$ .\venv\Scripts\activate
```

## 2. ACIプログラマビリティ

このモジュールでは、Cisco ACI SDNソリューションのプログラマビリティオプションを紹介し、ACIのプログラミングを開始する最も簡単な方法であるACIToolkitを実践的に提供します。

### 「ACIの理解」

アプリケーションセントリックインフラストラクチャとアプリケーションポリシーオブジェクトの基本を学びます。

### 「ACI プログラマビリティオプション」

APICコントローラとインターフェイスの基本を学び、CLIを使用して基本的なコマンドを発行します。

## 2.1. ACI の理解

### 目的

- NX-OSモードとACIの理解
- ACIのコンポーネントと用語の理解
- ACIテナントネットワーキングの理解
- ACIテナントポリシーの理解
- GUIを探索する

### 前提条件

これは入門ラボであり、前提条件はありません。

## ステップ 1: はじめに

### ACIモードとNX-OSモード

Nexus 9000には、さまざまな運用モデルに適合する2つの動作モードがあります。NX-OSモードとアプリケーションセントリックインフラストラクチャ(ACI)モードです。

NX-OSの運用モードでは、Nexus 9000プラットフォームは、拡張機能を備えた従来のオペレーティングシステムを利用して、次世代のデータセンタープロトコルとテクノロジーを活用する機能を維持しながら、既存のネットワーク展開用のプラットフォームを提供します。NX-OSモードでは、CiscoのNexus NX-APIなどのAPIを使用して、スタンドアロンまたはデバイスごとのプログラム可能性を実現できます。

ACIモードのCisco Nexus 9000では、インフラストラクチャはコントローラのクラスタであるアプリケーションポリシーインフラストラクチャコントローラ(APIC)によって集中管理されます。ACIは、データセンター向けのシスコのコアソフトウェア定義ネットワーク(SDN)ソリューションです。

NX-OSモードとACIモードには独立したロードマップと機能セットがありますが、共通のハードウェアプラットフォームは柔軟性、選択、および価値を提供します。参照されている一般的なハードウェアは、Nexus プラットフォーム、特にNexus 9000シリーズです。

### ACIハードウェア

ほとんどのNexus9000ハードウェアでは、ソフトウェアアップグレードを実行して、ハードウェアをSDNに重点を置いた運用モデルであるACIに移行できます。

ACIモードでは、スイッチはspine[スパイン]またはleaf[リーフ]として動作します。

- スパインスイッチは、リーフスイッチを集約するために使用されます。
- リーフスイッチはアクセスデバイスとして使用されます。

## ステップ 2 : ACI をさらに深く掘り下げる

ACIモードで実行されているNexusスイッチは、APICコントローラで動作しているオブジェクトベースのポリシーエンジンを介してのみプログラム可能です。コントローラはネットワークの一部として統合されており、スイッチを一元的にプログラミングするためのポリシーを含むプロファイルを保持しています。スイッチ自体は、NX-OSベースのシステムで以前に使用されていたCLI構成ファイルを維持しません。

構成は、オブジェクト指向スキーマを使用してAPICに保持されます。XMLまたはJSONのいずれかで表され、プロファイルに格納されて、アプリケーション主導、ネットワーク主導、およびセキュリティ主導のポリシーを実装します。

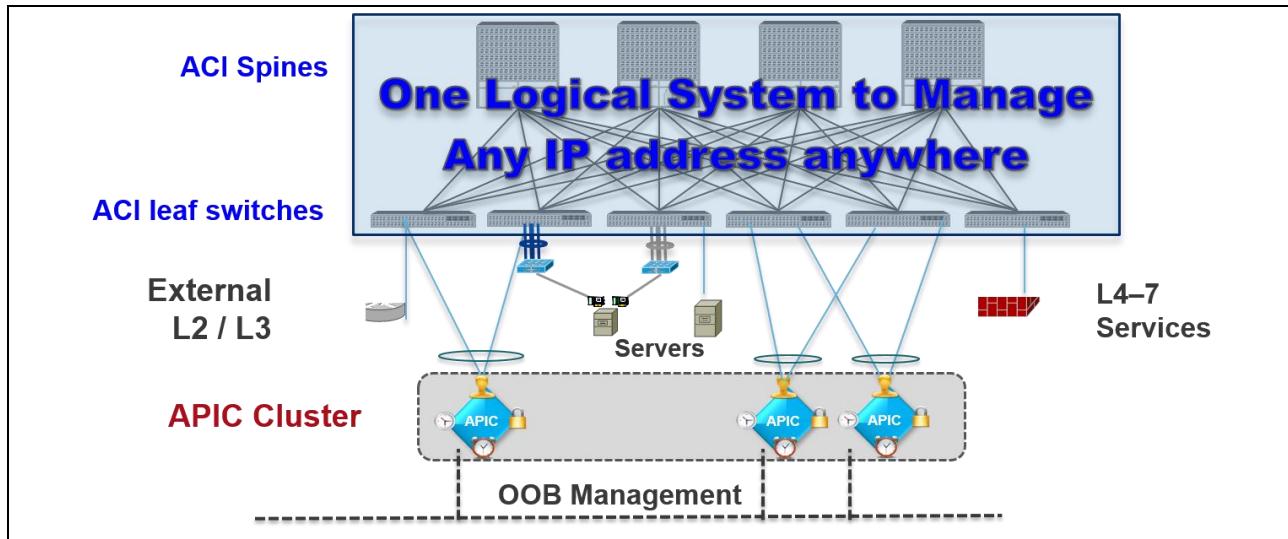


図1-7. ACIファブリック

### ACIコンポーネント

ACI展開のコアコンポーネントには、次のものがあります。

- **リーフスイッチ**は、Top-of-Rack(ToR)またはEnd-of\_Row(EoR)でファブリックへの接続を提供します。これらは、分散レイヤ3ゲートウェイ、ポリシー実施ポイント、および外部ネットワークへのゲートウェイとして機能します。
- **ボーダーリーフスイッチ**は、ファイアウォール、ロードバランサー、ルーター、非ACIスイッチなど、ACIファブリックの外部にあるネットワークデバイスに接続するリーフノードです。ACIネットワークへのスムーズな移行を可能にします。
- **スパインスイッチ**は、迅速な障害検出と再ルーティングを備えたノンブロッキングファブリックを提供します。これらは、2つのリーフスイッチ間でトラフィックを転送するために使用されます。ソフトウェアバージョン2.0(2)以降、ACIは、スパインスイッチへのイーサネットVPN(EVPN)を使用したレイヤ3接続をサポートします。
- **APICコントローラ**は、ファブリック構成の集中管理ポイントを提供し、動作状態の概要を監視します。ポリシーの観点からは、APICは構成の主要な連絡先であり、ポリシーリポジトリとして機能します。

### ステップ 3 : ACI の用語

APICはオブジェクトベースのポリシーエンジンを使用して動作することは前述しました。これにより、ネットワーク管理者はファブリックの望ましい状態を定義できますが、実装はコントローラー(APIC)に任せられます。ワークロードが移動すると、コントローラーは基盤となるインフラストラクチャを再構成して、エンドホストに必要なポリシーが引き続き適用されるようにします。これらのポリシーが定義されているオブジェクトモデルのプランチは、テナントオブジェクトです。これは、日常業務のほとんどが存在する場所です。

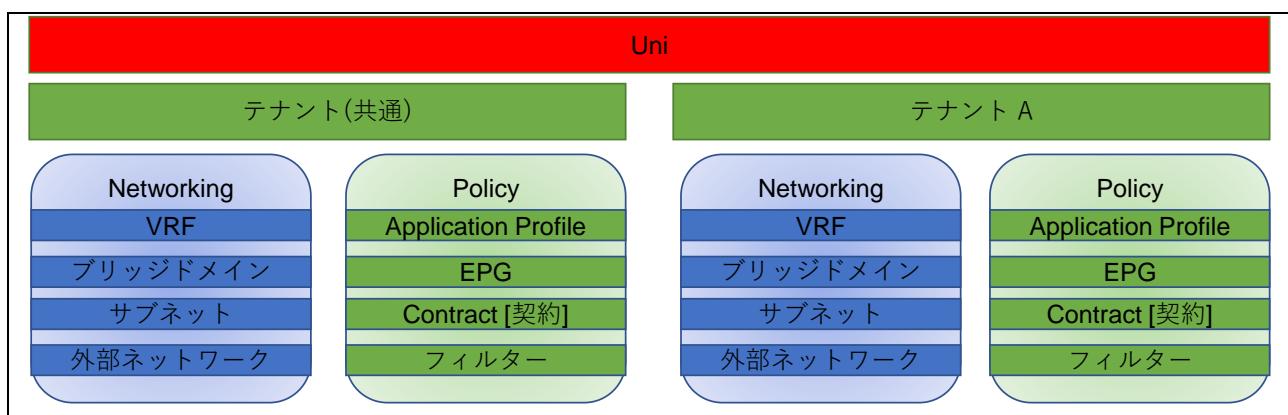


図1-8. テナントネットワークポリシー

### ACIテナント

テナントは、管理制御、ネットワーク/障害ドメイン、およびアプリケーションポリシーを識別して分離す

るトップレベルのオブジェクトです。テナントのサブレベルオブジェクトは、テナントネットワークとテナントポリシーの2つの基本的なカテゴリにグループ化できます。これらの2つのカテゴリには固有の関係がありますが、別々に説明すると役立つ場合があります。

## テナント「common[共通]」

ACIには、他のテナントとリソースを簡単に共有する独自の機能を備えた「common」という名前の特別なテナントがあります。「共通」テナントは、DNS、認証サービス、セキュリティツールなど、ACIファブリック全体に共有リソースを提供するように設計されています。「共通」テナントのすべてのテナントネットワークオブジェクト、コントラクト、およびフィルタを定義し、これらを他のテナントのアプリケーションに関連付けることができます。

## ステップ4：テナントネットワーキング

テナントネットワークオブジェクトは、エンジニアがすでに精通しているネットワーク構造に似ており、ホスト間のレイヤ2およびレイヤ3接続を提供します。テナントネットワークは、VRF(仮想ルーティングおよび転送)、ブリッジドメイン、サブネット、および外部ネットワークで構成されます。

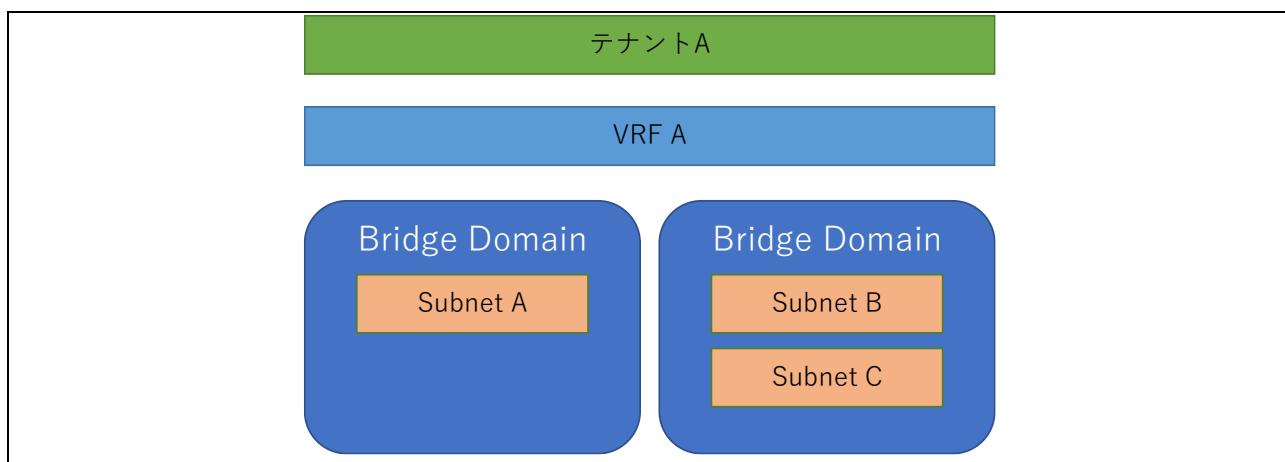


図1-9. テナントネットワーク

- コンテキストおよびプライベートネットワークとも呼ばれる**VRF**は、テナントの分離されたルーティングテーブルです。テナントは、1つまたは複数のVRFを持つことができます。または、「共通」テナントのVRFを使用することもできます。次の図では、**infra** VRFは灰色の領域にあります。追加のVRFはすべて、VRFにホストが割り当てられているリーフに存在します。
- **ブリッジドメイン**は、ファブリック内のレイヤ2転送ドメインであり、一意のMACアドレススペースとフラッディングドメイン(ブロードキャスト、不明なユニキャスト、およびマルチキャスト)を定義します。各ブリッジドメインは1つのVRFにのみ関連付けられますが、VRFは多くのブリッジドメインに関連付けることができます。VLANが従来展開してきた方法とは異なり、各ブリッジドメインには複数のサブネットを含めることができます。
- **サブネット**は、ホストがネットワークに接続するためのIPスペースとゲートウェイサービスを提供するレイヤ3ネットワークです。各サブネットは、1つのブリッジドメインにのみ関連付けられています。
- **外部ブリッジネットワーク**は、レイヤ2/スパニングツリーネットワークをACIファブリックに接続します。これは、従来のネットワークインフラストラクチャからACIネットワークにスムーズに移行するために、ブラウザフィールド環境で一般的に使用されます。次の図には、L4-L7デバイス用のレイヤ2外部ネットワークもあります。
- **外部ルーテッドネットワーク**は、ACIファブリックの外部のネットワークとのレイヤ3隣接関係を作成します。レイヤ3外部ネットワークは、スタティックルートまたはBGP、OSPF、およびEIGRPを使用した隣接関係をサポートします。レイヤ3接続には、コントラクトを提供および消費するネットワークも定義されています(テナントポリシーで詳しく説明します)。

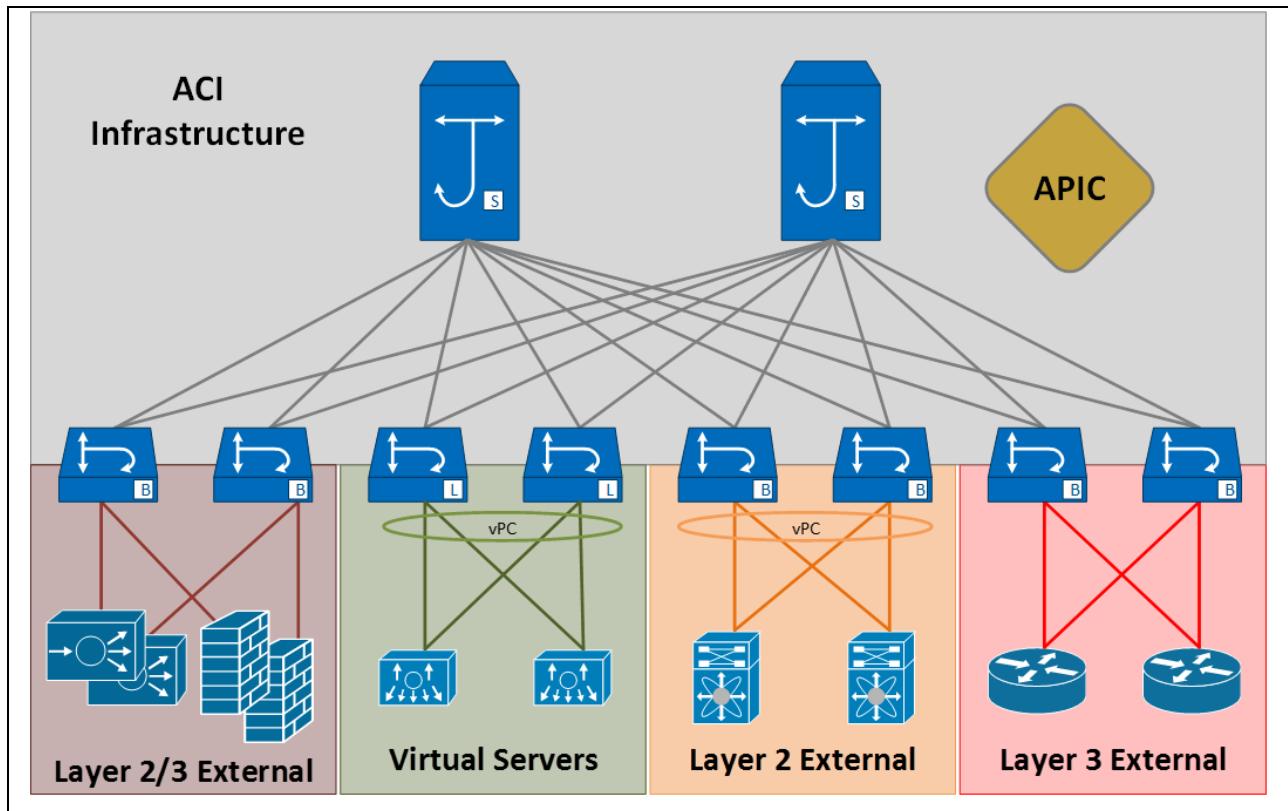


図1-10. 外部

## ステップ5：テナントポリシー

テナントポリシーオブジェクトはネットワークオブジェクトに関連していますが、テナントポリシーオブジェクトは、エンドポイントが受け取るポリシーとサービスにより重点を置いています。テナントポリシーは、アプリケーションプロファイル、エンドポイントグループ、コントラクト、およびフィルタで構成されます。これらはすべて新しいACI固有の用語です。

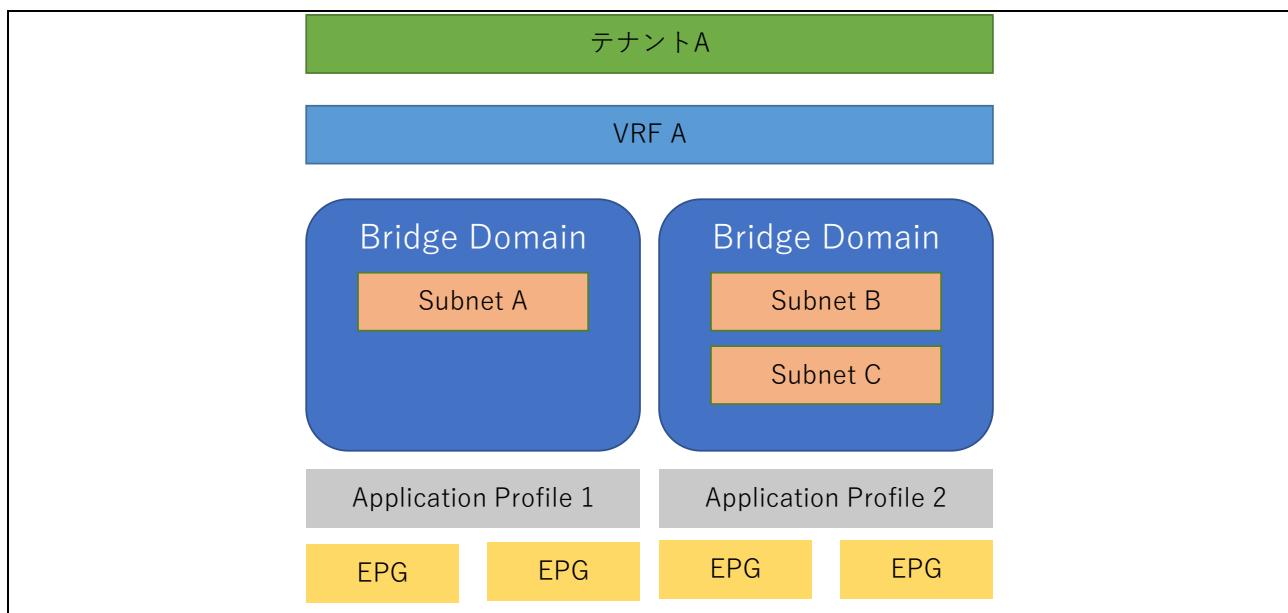


図1-11. テナントポリシー

- アプリケーションプロファイルは、エンドポイントグループ(EPG)のコンテナです。アプリケーションプロファイルは、アプリケーションの識別子として使用され、管理者権限の分離を可能にします。
- エンドポイントグループ(EPG)が適用され、同じサービスとポリシーを持っているエンドポイントのコレクションです。EPGは、アプリケーションサービスに関連付けられたスイッチポート、仮想スイッチ、およびレイヤ2カプセル化を定義します。各EPGは、1つのブリッジドメインにのみ関連付けることができます。

- **コントラクト(Contract)**は、EPGのエンドポイントに適用されるサービスとポリシーを定義します。コントラクトは、L4-L7デバイスへのサービスのリダイレクト、QoS値の割り当て、およびACLルールの適用に使用できます。サービスを提供するEPGがコントラクトを**提供**し、サービスを使用する必要があるEPGがコントラクトを**消費**します。
- **フィルタ**は、プロトコル(tcp、udp、icmpなど)とポートを定義するオブジェクトです。フィルタオブジェクトには複数のプロトコルとポートを含めることができ、コントラクトは複数のフィルタを使用できます。

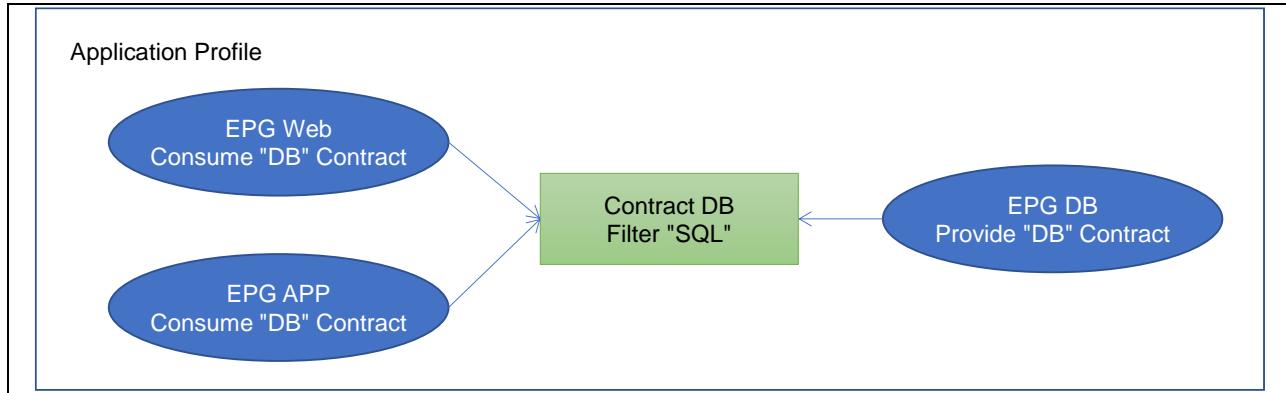


図1-12. ポリシーの例

最も一般的なテナントの概念を理解したので、APIC GUIでアプリケーションを構成しましょう。

## 演習 1 : ACI GUI ウォークスルー

「Heroes」という名前のテナントがAPIC GUIで構成されています。ウォークスルーを開始するには、APIC を参照してログインします。

APIC Login Information	
URL	<a href="https://apic1.dcloud.cisco.com">https://apic1.dcloud.cisco.com</a>
User	admin
Password	C1sco12345

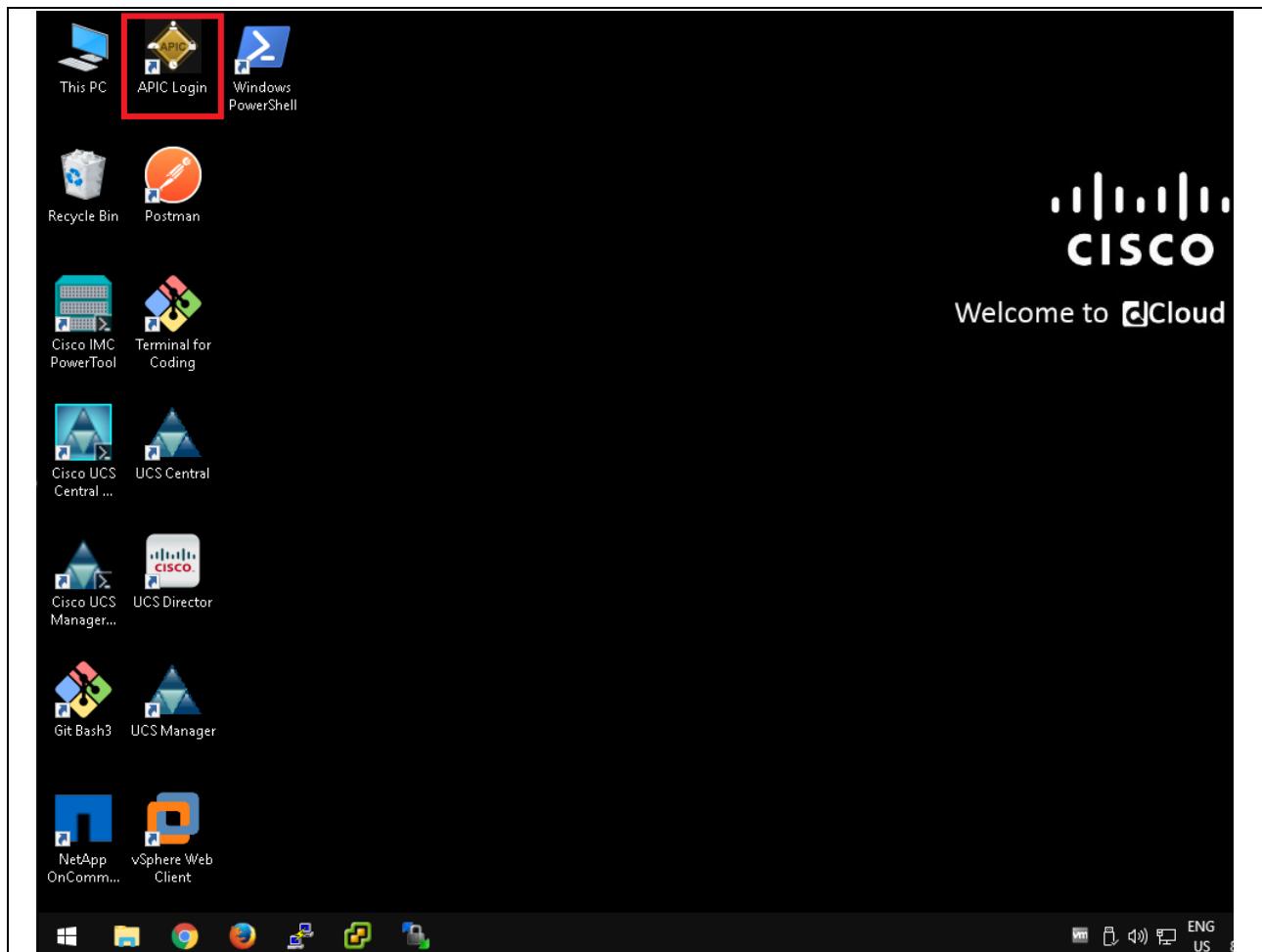


図1-13.

A screenshot of the Application Centric Infrastructure - ACI login page. The top header reads 'Application Centric Infrastructure - ACI'. Below it, the text 'Please sign in to connect to APIC' is displayed. There are two input fields: 'User ID: admin' and 'Password: .....'. A dropdown menu labeled 'Mode: Advanced' is shown below the password field. At the bottom is a blue 'LOGIN' button. The Cisco logo is in the bottom left corner. The background features a bar chart graphic on the right side.

図1-14. APICログイン

「Heros」テナントを確認する：

1. Tenants[テナント]をクリックします。
2. Heroesをダブルクリックします。

Name	Description	Bridge Domains	VRFs	EPGs	Health Score
common		1	2	0	100
Heroes		1	1	3	95
infra		1	1	1	100
mgmt		1	2	0	100
SnV		1	1	8	95

図1-15. テナント Herosを開く

アプリケーションプロファイルを作成する

新しいLayer2アプリケーションを作成し、既存のコントラクトを使用してサービスを提供および利用します。

GUIの場合：

1. Application Profiles[アプリケーションプロファイル]を右クリックします。
2. Create Application Profile[アプリケーションプロファイルの作成]を選択します。

図1-16. アプリケーションの作成

1. アプリケーションに「**自分の名前(イニシャル\_電話番号)\_Power\_Up**」という名前を付けます。  
例)jt\_1234\_Power\_Up
2. Submit[送信]をクリックします。

Create Application Profile

Specify Tenant Application Profile

Name: jt\_1234\_Power\_Up

Description: optional

Tags: enter tags separated by comma

Monitoring Policy: select a value

**EPGs**

Name	BD	Domain	Static Path	Static Path VLAN	Provided Contract	Consumed Contract

SUBMIT CANCEL

図1-17. アプリケーションの名前

### Web EPGを作成する

アプリケーションが作成されたので、このアプリケーションを形成する「Web」および「DB」EPGを構成します。

1. 「自分の名前\_Power\_Up」フォルダを展開して、EPGにアクセスします。
2. Application EPGsメニューから、Create Application EPG[アプリケーションEPGの作成]を選択します。

ALL TENANTS | Add Tenant | Search: enter name, descr | common | Heroes | infra | mgmt | SnV

Tenant Heroes

- Quick Start
- Tenant Heroes
  - Save\_The\_Planet
    - jt\_1234\_Power\_Up
      - Application EPGs
      - uSeg EPGs
      - L4-L7 Service Parameters
  - Networking
  - L4-L7 Service Parameters
  - Security Policies
  - Monitoring Policies
  - Troubleshoot Policies
  - L4-L7 Services

Application EPGs

Name	Description	QoS Class	Intra EPG Isolation
			No items have been found. Select Actions to create a new item.

ACTIONS -

Create Application EPG

図1-18. EPGを作成する

1. EPGに「Web」という名前を付けます。
1. Bridge Domain ドロップダウンからHeroes/Hero\_Landを選択します。
2. Finish[完了]をクリックします。

**Create Application EPG**

**STEP 1 > Identity**

Specify the EPG Identity

Name: Web
Description: optional
Tags: enter tags separated by comma
QoS class: Unspecified
Custom QoS: select a value
Intra EPG Isolation: Enforced Unenforced
Preferred Group Member: Exclude Include
Bridge Domain: Heroes/Hero_Land
Monitoring Policy: select a value
Associate to VM Domain Profiles: <input type="checkbox"/>
Statically Link with Leaves/Paths: <input type="checkbox"/>

PREVIOUS FINISH CANCEL

図1-19. Web EPGを作成する

Web層にコントラクトを提供させるには、次のようにします。

1. EPG Webを展開します。
2. Contracts[コントラクト]を右クリックします。
3. Add Provided Contract[提供されたコントラクトの追加]を選択します。

APIC (apic1.dcloud.cisco.com)

System Tenants Fabric VM Networking L4-L7 Services Admin Operations

Tenant Heroes Contracts

No items have been found.  
Select Actions to create a new item.

ACTIONS -

Tenant Name	Contract Name	Contract Type	Provided / Consumed	QoS Class	State	Label	Subject Label

Add Taboo Contract  
Add Provided Contract  
Add Consumed Contract  
Add Consumed Contract Interface

図1-20. Web EPGに提供されたコントラクトの追加

Add Provided Contract [提供されたコントラクトの追加]ウインドウから :

1. Contracts[コントラクト]ドロップダウンメニューから[common/web]を選択します。

2. Submit[送信]をクリックします。

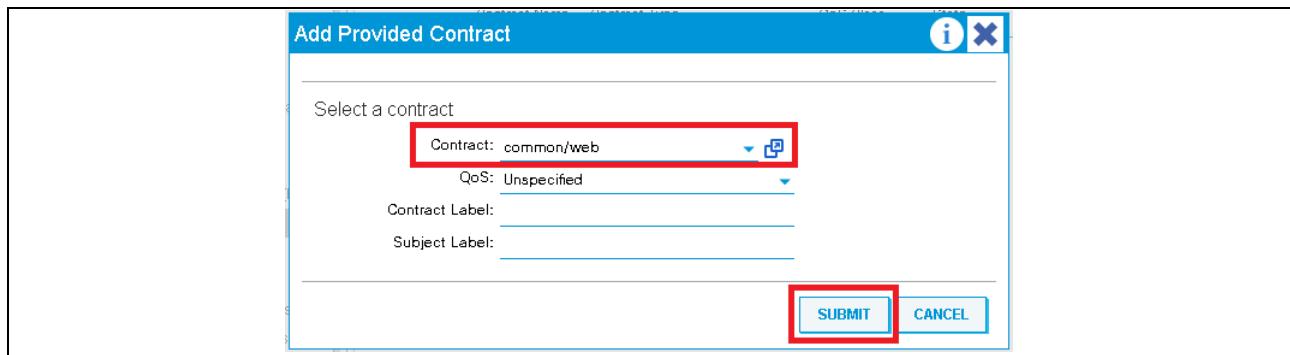


図1-21. Contractの選択

コントラクトの消費は同じ手順に従います

1. 今度はAdd Consumed Contract [消費コントラクトの追加]を選択します。

The screenshot shows the Cisco ACI interface under the 'Tenants' tab. On the left, there's a navigation tree for 'Tenant Heroes' with various EPGs and contracts. In the center, a table lists contracts with columns: Tenant Name, Contract Name, Contract Type, Provided / Consumed, QoS Class, State, Label, and Subject Label. One row is shown: 'common web Contract Provided Unspecified formed'. On the right, a context menu is open over the 'Contracts' list, with the 'Add Consumed Contract' option highlighted by a red box.

図1-22. SQLを消費する

2. Contract [コントラクト]ドロップダウンメニューからcommon/sqlを選択します。

3. Submit [送信]をクリックします。

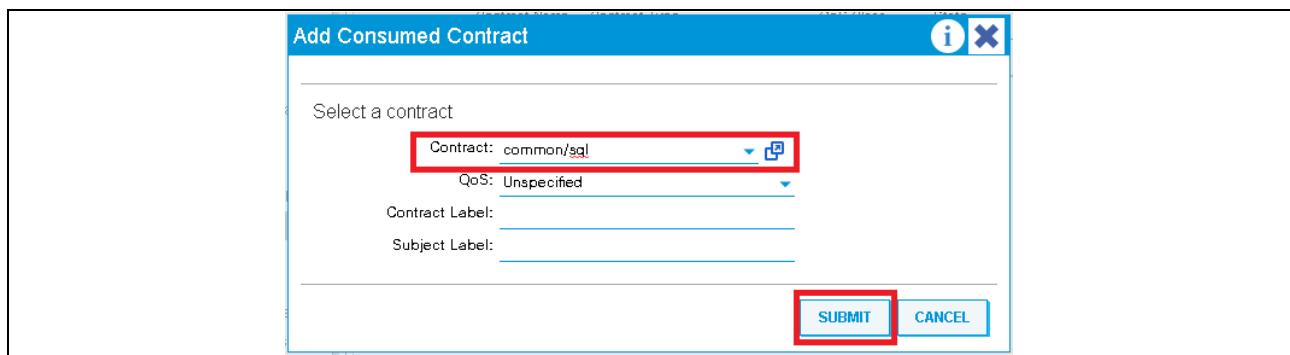


図1-23. Consumed Contractの選択

Web EPGが作成されます。同じ手順に従って、EPG 「DB」を作成します。

#### EPG DBを作成する

次の手順で「DB」EPG 「DB」を作成します。

1. Application EPGs [アプリケーションEPG]を右クリックします。

## 2. Create Application EPG [アプリケーションEPGの作成]を選択します。

The screenshot shows the Cisco Application Policy Infrastructure (ACI) interface. In the left sidebar under 'Tenant Heroes', 'Application EPGs' is selected, and a red box highlights the '+ Create Application EPG' button. The main pane displays a table of existing EPGs with columns for Name, Description, QoS Class, and Intra EPG Isolation.

Name	Description	QoS Class	Intra EPG Isolation
Web		Unspecified	Unenforced

図1-24. EPGを作成する

Create Application EPG [アプリケーションEPGの作成]ウィンドウで：

1. EPGに「DB」という名前を付けます。
2. ブリッジドメインからドロップダウンして、Heros/Hero\_Landを選択します。
3. Finish[完了]をクリックします。

The screenshot shows the 'Create Application EPG' configuration window. The 'Identity' tab is selected. The 'Name' field is set to 'DB'. The 'Bridge Domain' dropdown is set to 'Heroes/Hero\_Land'. The 'FINISH' button at the bottom right is highlighted with a red box.

図1-25. DBを作成する

DB層に「SQL」コントラクトを提供させます。

1. EPG DBを展開します。

2. Contract[コントラクト]を右クリックします。
3. Add Provided Contract[提供されたコントラクトの追加]を選択します。

The screenshot shows the Cisco ACI interface. In the top navigation bar, 'Tenants' is selected. On the left, under 'Tenant Heroes', there's a tree view with 'Contracts' expanded. A context menu is open over the 'Contracts' link, with 'Add Provided Contract' highlighted. The main pane displays a table of contracts:

Tenant Name	Contract Name	Contract Type	Provided / Consumed	QoS Class	State	Label	Subject Label
common	sql	Contract	Consumed	Unspecified	formed		
common	web	Contract	Provided	Unspecified	formed		

図1-26. DB提供

Add Provided Contract [提供されたコントラクトの追加]ウィンドウで：

1. Contract[コントラクト]ドロップダウンメニューから[common/sql]を選択します。
2. Submit[送信]をクリックします。

The screenshot shows the 'Add Provided Contract' dialog box. It has fields for 'Contract' (set to 'common/sql'), 'QoS' (set to 'Unspecified'), 'Contract Label', and 'Subject Label'. At the bottom are 'SUBMIT' and 'CANCEL' buttons.

図1-27. SQLを提供する

DB層はサービスを消費しません。これで、SQLサービスのDBにアクセスできるWebサーバーがアプリケーションに完成しました。「Common」テナントからの既存のコントラクトが使用されたため、必要なリソースはWebEPGからのWebサービスを利用することもできます。

ACIのプログラミングを開始する前に、テナントネットワークとテナントポリシーを理解することが非常に重要です。

### お疲れ様でした!

基本がカバーされたので、ACIが提供しなければならない様々なプログラマビリティオプションを見てみましょう。

## 2.2. ACIプログラマビリティオプション

### 目的

- ACIオブジェクトモデルと管理情報ツリーを理解する
- 相対名と識別名を理解する
- ACIプログラマビリティオプションの概要
  - REST API
  - ACIツールキット
  - Cobra SDK
  - Aryaおよびその他のツール

### 演習 0: APIC サンドボックス環境のセットアップ<sup>①</sup>

1. Windows Powershellを開き、ディレクトリを変更します。(ディレクトリを間違えないように入力してください)

```
> cd aci-learning-labs-code-samples\apic_fabric_setup
```

2. credentials.pyファイルを編集します。

ファイルの場所: c:\Users\demouser\aci-learning-labs-code-samples\apic\_fabric\_setup\credentials.py

```
URL = 'https://apic1.dcloud.cisco.com'  
LOGIN = 'admin'  
PASSWORD = 'Cisco12345'
```

3. 次のコードリポジトリにあるベースラインスクリプトを実行します。

```
apic_fabric_setup/baseline.py  
> python baseline.py
```

# 期待される出力

```
Baselining APIC Simulator for Learning Labs  
Setting up Fabric Nodes  
Configuring Fabric Policies  
Setting up Common Tenant  
Setting up Heroes Tenant  
Setting up SnV Tenant
```

### ステップ 1：はじめに

#### なぜプログラマビリティのためのACI？

ACIのプログラマビリティオプションを確認する前に、ネットワークのプログラマビリティの現在の状態を確認しましょう。

ネットワークは従来、デバイスごとに構成および保守されるように設計されたデバイスで構築されてきました。変更やトラブルシューティングを行うには、ネットワークエンジニアは複数のデバイスに個別に接続し、CLIでコマンドを入力する必要があります。このソリューションは、ほとんど静的な環境で正常に機能しますが、拡張性がなく、変更がより頻繁に繰り返し行われるため、人的エラーが発生しやすくなります。CLIは人間向けに構築されているため、プログラマビリティと自動化のための理想的なインターフェイスとは言えません。

これは、ACIがデータセンターの運用に関して解決する主要な問題です。ACIのプログラマビリティオプションは、ACIオブジェクトモデルを通じて可能になります。

### オブジェクトモデル

「ACIの理解」では、ACIは、ACIの構成と管理に使用されるオブジェクトベースのモデルを使用して動作すると述べられています。ACIと対話するプログラムによる方法を検討する前に、このオブジェクトモデルがどのように機能するかについての基本的な理解が必要です。オブジェクトモデルは、論理と具体的な2つのカテゴリに分類できます。

管理者と管理ツールは、Application Policy Infrastructure Controller(APIC)を介して論理モデルと対話します。論理モデルに加えられた変更は、ハードウェアとソフトウェアが必要に応じてプログラムされる具体的なモデルにプッシュダウンされます。論理と具象を分離すると、デバイスごとにネットワークを管理するのではなく、単一の管理インターフェイスを作成して目的の状態を定義できます。単一の管理インターフェイスは、プログラミングおよび自動化戦略を簡素化するのに役立ちます。

例：管理者がEPGを作成し、VLAN10を割り当てます。管理者は論理モデルを使用しています。次に、論理モデ

ルは、EPGにホストがあるスイッチでのみVLAN 10を構成することにより、VLAN10を具象モデルにプッシュします。

管理者は論理モデルで作業するため、このモデルがどのように機能するかについて説明します。論理モデルはオブジェクトの階層として編成され、ルートが最上位のオブジェクトです。

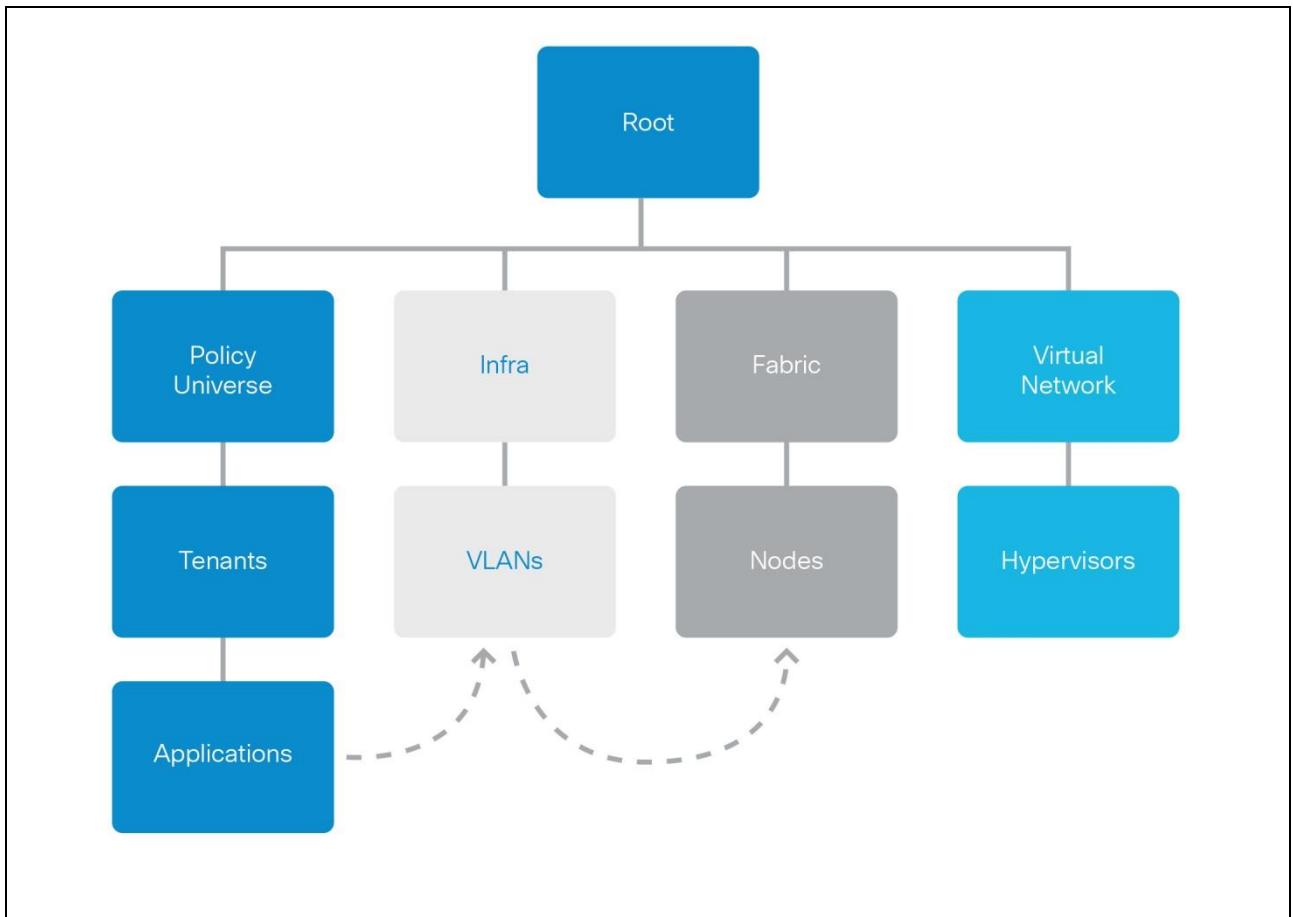


図1-28. オブジェクトモデル

## ステップ 2：管理対象オブジェクト

ACIファブリック内のすべてがオブジェクトです。これらのオブジェクトは管理対象オブジェクトと呼ばれ、MOと呼ばれることがよくあります。各MOは、それが持つプロパティとその機能を決定する特定のクラスに属しています。テナントオブジェクトの機能の1つは、APIC GUIのクラスへの管理アクセスを制限することです。

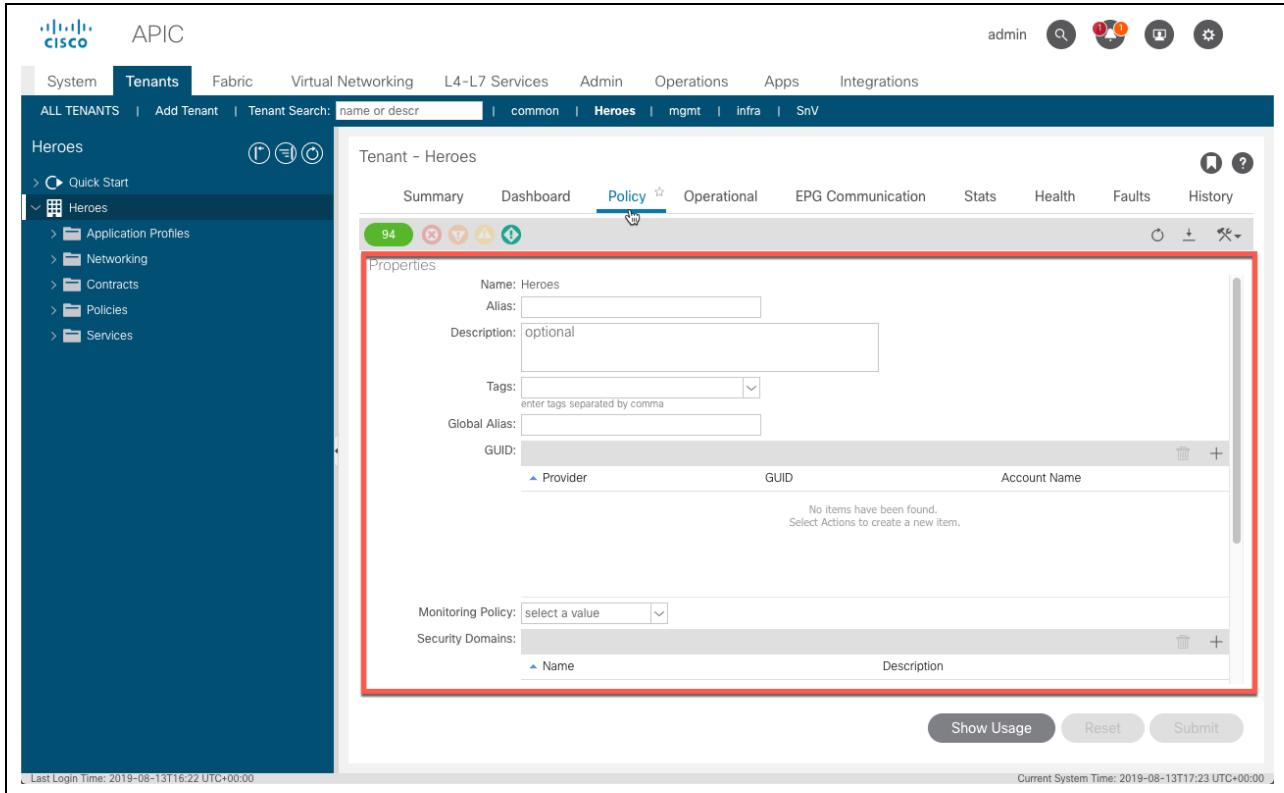


図1-29. テナントオブジェクト

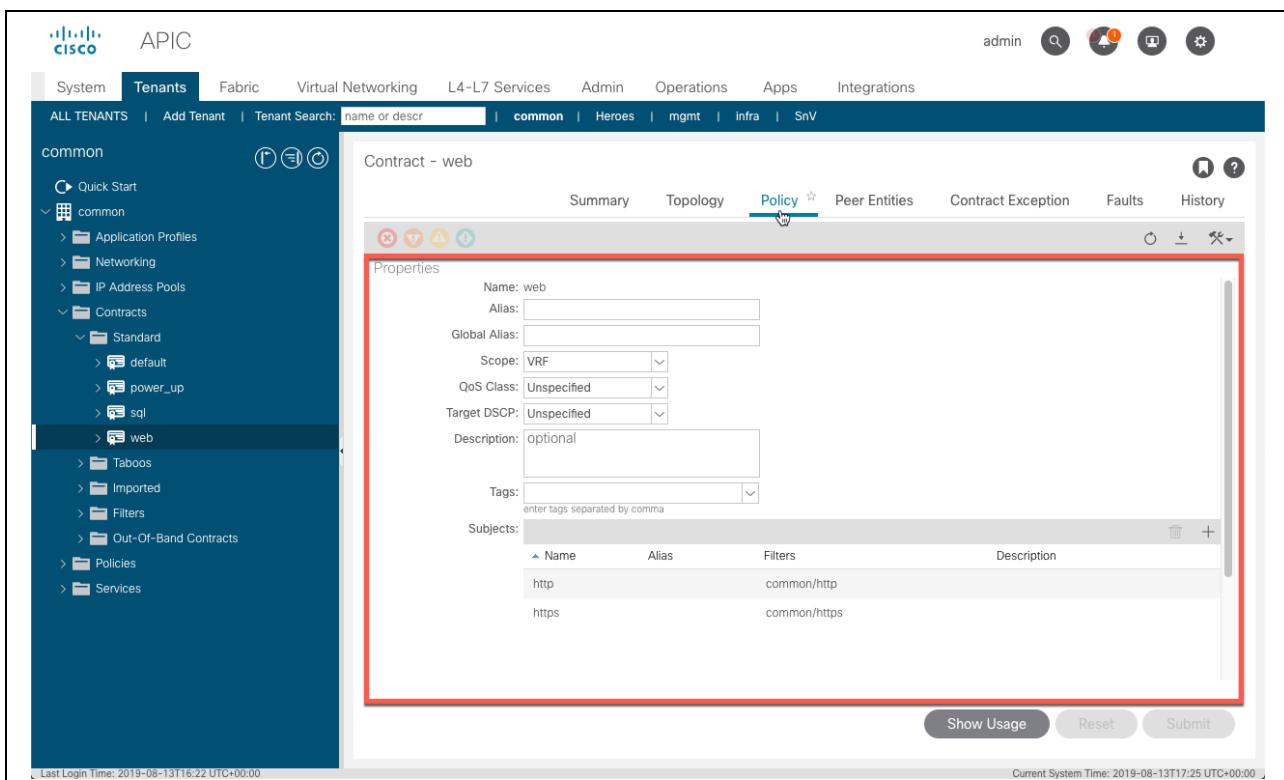


図1-30. コンtractオブジェクト

### ステップ 3：管理情報ツリー

管理対象オブジェクト(MO)は、管理情報ツリー(MIT)を形成する親子関係で接続されます。ルートを除いて、すべてのMOには親があります。MITは、オブジェクトの親をルートまで参照することにより、オブジェクトを追加、削除、または照会するために使用されます。オブジェクトは、親と子のパラダイムの外で関係を持つこともできます。次の例では、**EPG**アプリオブジェクトは**Save\_The\_Planet**アプリケーションプロファイルの子ですが、**Bridge Domain**および**Contracts**オブジェクトとも関係があります。

図1-31. オブジェクトの関係

関係は、1対1、1対多、または多対多の場合があります。次の図は、テナントツリーの一部を示し、オブジェクトを関連付けるさまざまな方法を示しています。

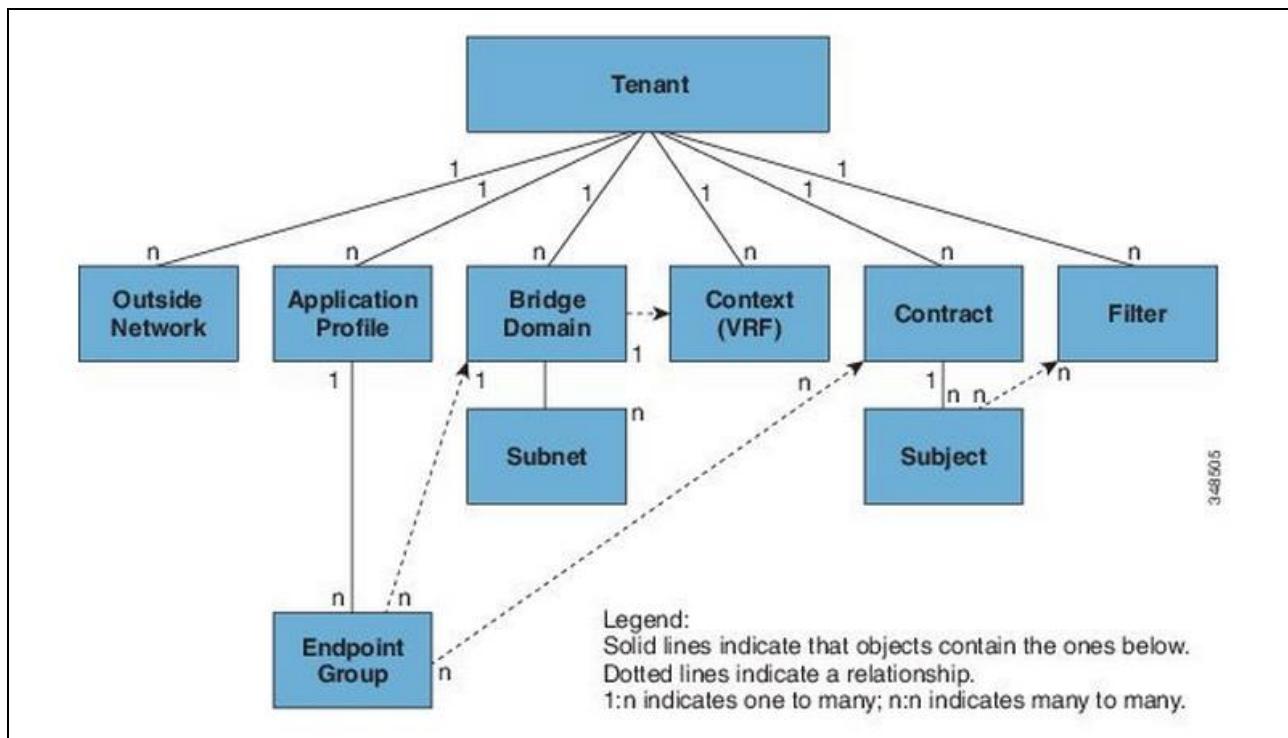


図1-32. MITの例

これらの関係は、関係に含まれるオブジェクトが使用できるリソースを決定するため、重要です。次の図は、これらの依存関係を示す具体的な例です。

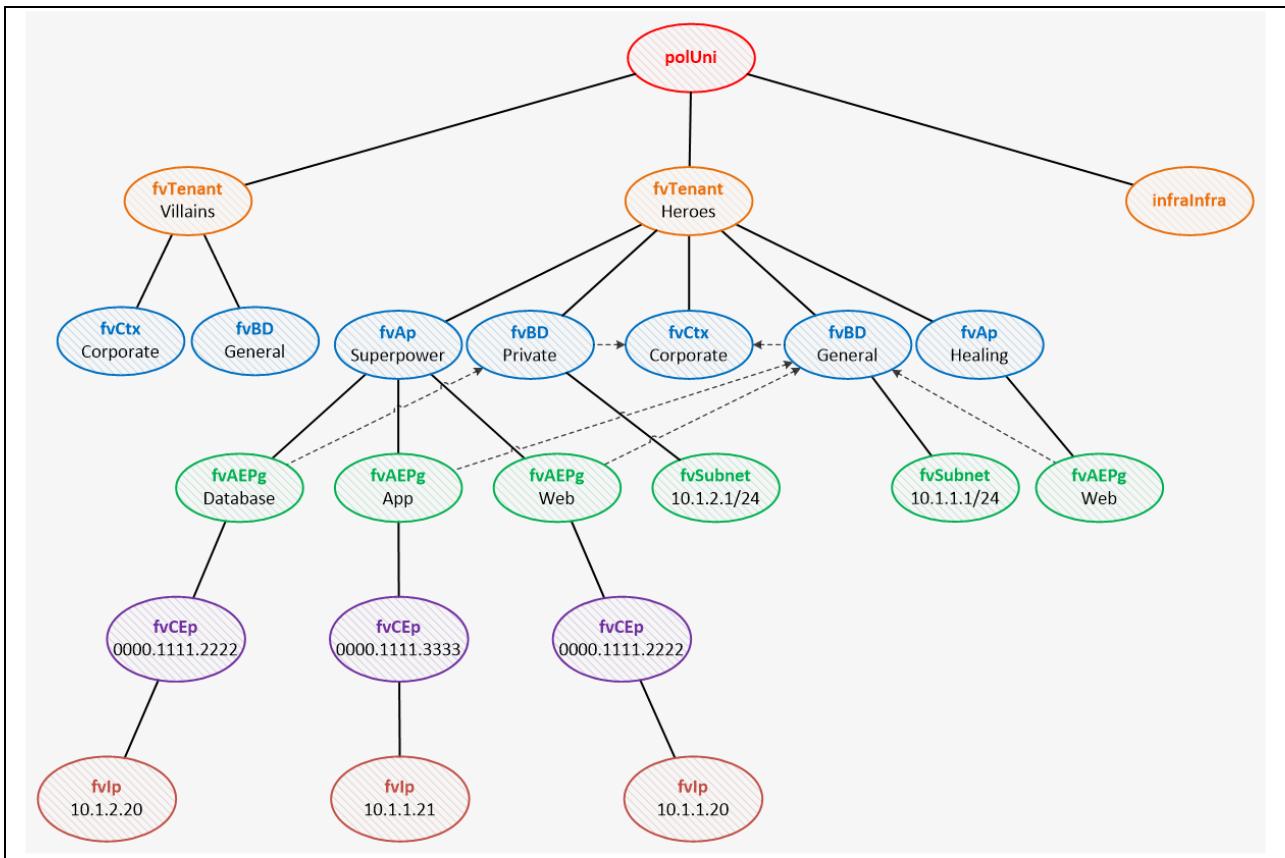


図1-33. MITオブジェクト

この図から注意すべき点がいくつかあります。

- オブジェクトは、親オブジェクトに関連して一意に識別できます。
  - EPG(fvAEPg)Webは2回使用されますが、親オブジェクトが異なるため、一意です。
- 両方のブリッジドメイン(fvBD)が同じVRF(fvCtx)に属しているため、ヒーローのすべてのIPアドレス(fvIp)は一意である必要があります。
- エンドポイント(fvCEp)が異なるブリッジドメインに関連するEPGに属している限り、MACアドレスを複製できます。
  - Superpower/Database/0000.1111.2222はfvBDPrivateとの関係を形成します。
  - Superpower/Web/0000.1111.2222はfvBDGeneralとの関係を形成します。
- EPG /ブリッジドメインの関係により、エンドポイントが使用できるサブネット(fvSubnet)が決まります
  - Superpower/Database/0000.1111.2222は10.1.2.0/24のIPを使用する必要があります
  - 他のすべてのEPGは、10.1.1.0/24のIPを使用する必要があります

プログラミングに関するMITのこの議論から収集する3つのことは次のとおりです。

- 構成および統計データには、MOを参照してアクセスします。
- オブジェクトにアクセス/作成するには、ルートから開始し、ツリーを下って一意のオブジェクトに移動します。
- プログラムは、オブジェクト間の関係を考慮する必要があります。
  - 関数またはプログラムを使用する場合、どのような引数を含める必要がありますか？

たとえば、EPGを作成するには、それをMITに正しく配置するために、テナントオブジェクトとアプリケーションオブジェクトが必要です。また、エンドポイントを正しいサブネットとVRFに接続するためのブリッジドメインも必要です。

- オブジェクトを削除する前に、どのようなテストを実行する必要がありますか？
- たとえば、ブリッジドメイン(BD)を削除する前に、EPGがBDと関係があるかどうかを確認してください。

## ステップ 4：相対名

各MOには、同じ親の他の子オブジェクトからオブジェクトを識別する**相対名(RN)**があります。各RNは親オブジェクト内で一意である必要がありますが、別の親で複製することができます。

図1-34. 重複したRN

相対名は**クラスプレフィックス**で始まり、オブジェクトのクラスタイプに基づいています。RNの最後の部分は、クラスがオブジェクトを識別するために使用するプロパティ識別子(通常は「名前」プロパティ)から取得されます。EPGプレフィックスは「epg-」で、プロパティ識別子は「name」です。前のツリー図を見ると、両方の「Web」EPGは同じRN「epg-Web」を持っています。一般的に使用されるプレフィックスとプロパティIDの表を次に示します。

一般名	プレフィックス- モジュール プロパティ	クラス	親クラス	例	
Tenant	tn-name	fv	Tenant	Uni	tn-Heros
Context/VRF	ctx-name	fv	Ctx	Tenant	ctx-Corporate
ブリッジドメイン	BD-name	fv	BD	Tenant	BD-General
サブネット	subnet-ip	fv	Subnet	BD	subnet-10.1.2.1/24
アプリプロファイル	ap-name	fv	Ap	Tenant	ap-Superpower
EPG	egg-name	fv	AEPg	Ap	epg-Database
クライアントエンド ポイント	cep-name	fv	CEp	AEPg	cep-0000.1111.2222
IP Address	ip-addr	fv	IP	CEp	ip-10.1.2.20
L3 External	out-name	l3ext	Out	Tenant	out-Corporate
Filter	fit-name	vz	Filter	Tenant	fit-HTTP
Contract	brc-name	vz	BrCP	Tenant	brc-Web_Services
Contract Subject	subj-name	vz	Subj	BrCP	subj-HTTP

Module列とClass列はデータモデルによって使用され、APIとCobra SDKの両方に表示されます。

現在親オブジェクトの下にいる場合は、RNを使用してオブジェクトにアクセスできます。これらは、API クエリの結果をフィルタリングするために頻繁に使用されます。

たとえば、アプリケーションの所有者が特定のホストに関する問題を報告します。この問題を調査するために、ネットワーク管理者はfvCEpクラスにクエリを実行し、子fvIpが10.1.1.20(ip-10.1.1.20)のオブジェクトのみを

返すことができます。

## 識別名

MITでは、親オブジェクトを参照することでオブジェクトを識別できます。これは、オブジェクトの**識別名**(DN)です。定義上、MIT内のすべてのオブジェクトには一意のDNがあります。

DNは、rootで始まり、各子RNをオブジェクトにアタッチする一連のRNで構成されます。

dn = root/{rn}/{rn}/{rn}

上記のツリー図では、2つのEPGのRNは同じですが、DNは異なります。

- uni/tn-Heroes/ap-Save\_The\_Planet/epg-Web
- uni/tn-Heroes/ap-Healing/epg-Web

通常、オブジェクトのIDから始めて、逆にDNを構築する必要があります。これは、特定のサービスに関連付けられたエンドポイントを見つけるのに役立つ「Web」EPGのDNを構築する例です。複数の「Web」EPGがあるため、正しい結果を得るには、適切なDNを構築することが重要です。この例では、アプリケーションはSuperpowerです。

1. オブジェクトRNをビルドします : **epg-Web**
2. ルートへの親オブジェクトを再帰的にビルドします。
  - fvApは親クラスであり、「Superpower」は特定のfvApオブジェクトです : **ap-Superpower/epg-Web**
  - fvTenantは次の親クラスであり、「Heroes」は特定のfvTenantオブジェクトです。  
**tn-Heroes/ap-Superpower/epg-Web**
  - ルートはfvTenantの親です : **uni/tn-Heroes/ap-Superpower/epg-Web**
3. 完全なDNは**uni/tn-Heroes/ap-Superpower/epg-Web**です。

DNは、特定のオブジェクトに対するAPI要求を行うために使用されます。

たとえば、アプリケーションSave\_The\_Planetの障害とログデータを表示するために、APIリクエストはリクエストURLの一部として一意のDN **uni/tn-Heroes/ap-Save\_The\_Planet**を使用します。

オブジェクトモデルの概要がわかったので、次に存在するさまざまなACIプログラマビリティオプションを見てみましょう。

## 演習 1 : ACI REST API

ACIは、プログラマビリティを念頭に置いて構築され、RESTAPIを介して中央コントローラを介して構成および保守されるように設計されています。このAPIは、管理者がオブジェクトモデルと対話して、ACIファブリックの作成、変更、統計の収集、およびトラブルシューティングを可能にする方法です。

POSTMANを使用すると、RNとDNについて現在知っていることを使用して、Create、Read、Update、およびDeleteメソッドを実行できます。

### APICログイン

ACI API中級ラボでは、APIの使用について詳しく説明します。この演習では、ログインと構成の変更でHTTP POSTが使用され、データの読み取りでHTTPGETが使用されることを知っておく必要があります。

1. POSTMANを使用してAPICにログインします
2. メソッドをPOSTに設定します
3. 下記のJSONを入力ボディとしてraw
4. [送信]をクリックします

APIC API	
メソッド	POST
	https://apic1.dcloud.cisco.com/api/aaaLogin.json

Body / raw JSON	<pre>{     "aaaUser": {         "attributes": {             "name": "admin",             "pwd": "C1sco12345"         }     } }</pre>
-----------------	--

200 OKが返されることを確認してください。

The screenshot shows the Postman interface with the following details:

- Request URL:** https://apic1.dcloud.cisco.com/api/aaaLogin.json
- Method:** POST
- Headers:** (9 items)
- Body:** (highlighted with a red box)
  - Raw dropdown: raw
  - JSON dropdown: JSON (highlighted with a red box)
  - Content: A JSON object representing user credentials.

```

1 {
2     "aaaUser": {
3         "attributes": {
4             "name": "admin",
5             "pwd": "C1sco12345"
6         }
7     }
8 }
```
- Response:** 200 OK 35 ms 1.96 KB
- ResponseBody:** (highlighted with a red box)
  - Pretty: {
  - Raw: "totalCount": "1", "imdata": [ { "aaaLogin": { "attributes": { "token": "BgR0vbZFHFDw9blWxJ03cIufAv/903TYwJK1710E9g6paaIvmIt..."} } ] }
  - Preview: (JSON structure shown)
  - Visualize: (graphical visualization)
  - JSON dropdown: JSON

図1-35. Postmanログイン

## READ

読み取り要求を行うには、HTTPメソッドをGETに変更します。

ACIファブリックで構成されているすべてのアプリケーションに対してクラスクエリを作成するには、次のURLを使用し、[送信]を選択して応答データを表示します。URLは、アプリケーションプロファイルのモジュール名とクラス名を組み合わせてクエリを実行することに注意してください。

メソッド	GET

URL	<a href="https://apic1.dcloud.cisco.com/api/class/fvAp.json?">https://apic1.dcloud.cisco.com/api/class/fvAp.json?</a>
-----	---

The screenshot shows the Postman interface with a red box highlighting the URL field containing the API endpoint. Below the URL, the status bar indicates a successful '200 OK' response. The main body area displays the JSON response, which includes a total count of 7 items and a detailed list of application profile objects (fvAp) with their attributes like dn, modTs, and name.

```

1 "totalCount": "7",
2 "imdata": [
3   {
4     "fvAp": {
5       "attributes": {
6         "childAction": "",
7         "descr": "",
8         "dn": "uni/tn-infra/ap-access",
9         "lcOwn": "local",
10        "modTs": "2021-08-26T06:39:07.638+00:00",
11        "monPolDn": "uni/tn-common/monepg-default",
12        "name": "access"
13      }
14    }
15  ]
16 ]
17 
```

図1-36. Postman AppClassクエリ

応答本文には、すべてのアプリケーションプロファイルが一覧表示され、各属性の辞書が提供されます。

## REST APIのユビキタス

POSTMANは、RESTAPIとの対話に使用されました。ただし、この同じAPIは、以下を含むAPICと通信するすべてのインターフェイスで使用されます。

- GUI
- CLI
- ACIツールキット
- Cobra SDK

このラボの残りの部分では、ACIToolkitとCobraを紹介します。将来のラボでは、これらのツールやその他のツールを使用してAPIを操作する必要があります。

## 演習 2 : ACI ツールキットを使用する

### ターミナルとコードノート

この演習では、Pythonのコーディング端末を開き、このラボのコードサンプルのディレクトリを入力します。

> aci-learning-labs-code-samples/sbx-intro-aci/sbx-intro-aci-02\_programmability-options

### ACIツールキット

多くの人がPOSTMANのようなツールでREST APIを使い始めます。これは、テストAPI呼び出しに慣れて行う簡単な方法です。ACIは、APICとの対話をプログラムで簡単に開始できる別のツールであるACIToolkitを提供します。ACI Toolkitは、MITでより一般的に使用されるオブジェクトのサブセットと対話するために構築されたPythonライブラリのセットです。また、特定のタスクを実行するために構築された既製のPythonアプリのコレクションも付属しています。

ACI Toolkitは、テナントクラス内のほとんどのオブジェクトを作成、読み取り、更新、および削除するために使用でき、ファブリックアクセスポリシー(レイヤ2カプセル化、物理ポート、ポートチャネル/ vpcなど)とも連携します。 )。

### 資格情報ファイル

**credentials.py**ファイルがAPICでの認証に、このラボで使用されています。このファイルは、このラボのコードサンプルディレクトリにあります。

## 1. credentials.pyファイルを編集します。

ファイルの場所:c:\Users\demouser\aci-learning-labs-code-samples\sbx-intro-aci\sbx-intro-aci-02\_programmability-options\credentials.py

```
URL = 'https://apic1.dcloud.cisco.com'  
LOGIN = 'admin'  
PASSWORD = 'C1sco12345'
```

## ツールキットテナントを作成する

ACI Toolkitラボでは、ACI Toolkitの使用方法について詳しく説明します。今のところ、それを紹介するテナントを作成します。

- Pythonインタラクションを開き、次のコードスニペットを参照してください。アクティビ化されたPython3仮想環境を使用する必要があるためpython、コマンドプロンプトで入力するだけです。

```
from credentials import *  
from acitoolkit import acitoolkit  
  
# disable Secure Request Warning  
import urllib3  
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)  
  
# connect to the apic  
session = acitoolkit.Session(URL, LOGIN, PASSWORD)  
session.login()  
# <Response [200]> --> If you do not see this response the login did not work  
  
# Create a Variable for your Tenant Name  
# Use your initials in the name  
# Example: tenant_name = "js_Toolkit_Tenant"  
tenant_name = "自分の名前_Toolkit_Tenant"  
  
# create a new tenant  
new_tenant = acitoolkit.Tenant(tenant_name)  
  
# commit the new configuration  
session.push_to_apic(new_tenant.get_url(), new_tenant.get_json())  
# <Response [200]> --> If you do not see this response the tenant create did not work
```

これは単純な例ですが、ACI Toolkitを使用して、複数のEPGを使用して新しいアプリケーションを構築し、EPGを適切なブリッジドメインに割り当て、コントラクトを提供および消費し、物理インフラストラクチャと仮想インフラストラクチャに接続するスクリプトを作成できます。また、視覚化を提供し、エンドポイントの場所やファブリック構成などの永続データを保存する既製のアプリケーションもいくつかあります。次のラボでは、ツールキットの機能について詳しく説明します。

ACI Toolkitは、ファブリックに対してプログラミングするための優れた方法ですが、ファブリックに対して最も一般的な操作のみを実行するように構築されています。Cobra SDK Pythonライブラリは、MITの完全に機能する1:1マッピングです。

## 演習 3：Cobra SDK を使用する

### ターミナルとコードノート

この演習では、コーディングターミナルを開き、このラボのコードサンプルのディレクトリを入力します。

```
> aci-learning-labs/sbx-intro-aci/sbx-intro-aci-02_programmability-options
```

### Cobra SDK

Cobraは、ACIファブリックのCRUD操作をサポートするPython SDKです。これはオブジェクトモデルの完全なマッピングであるため、ACIツールキットライブラリよりも複雑です。

オブジェクトモデルの完全なマッピングとは、次のことを意味します。

- 完全なMITはCobra SDKに組み込まれています。
- APIを介してアクセスできるすべての操作は、Cobraを介して実行することもできます。
- SDKはMITをモデルにしているため、Pythonのクラス名とプロパティ名はMITの名前と一致します。
  - Cobra ACIモジュールは、最初の大文字までの文字です。
  - CobraクラスはIDの残りの部分です。

- 例：  
fvTenant : モジュール= fv、クラス=テナント  
l3extOut : モジュール= l3ext、クラス=出力
- プロパティには、オブジェクトで「.property」を呼び出すことでアクセスできます。
- 例：  
tenant.name  
endpoint.ip

Cobraは完全な機能を提供するため、フィルタを使用したより複雑なクエリ、ACI外部のネットワークサービスデバイス用のL4-L7デバイスパッケージの組み込み、snmp、syslog、spineBGPポリシーなどのあまり一般的でないタスクを構成する初期ファブリックビルドスクリプトの作成に適しています。

## Cobra テナントを作成する

ACI Toolkitの場合と同様に、新しいテナントを作成することでCobraを紹介します。

新しいPythonインタプリタセッションを開き、次のコードスニペットを参照してください。

```
from credentials import *
import cobra.mit.access
import cobra.mit.request
import cobra.mit.session
import cobra.model.fv
import cobra.model.pol

#disable Secure Request Warning
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

# connect to the apic
auth = cobra.mit.session.LoginSession(URL, LOGIN, PASSWORD)
session = cobra.mit.access.MoDirectory(auth)
session.login()
# <Response [200]> --> If you do not see this response the login did not work

# 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
root = cobra.model.pol.Uni('')
new_tenant = cobra.model.fv.Tenant(root, tenant_name)

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

# <Response [200]> --> If you do not see this response the tenant create did not work
```

ツールキットがCobraの使用に伴う複雑さのいくつかを抽象化していることがすぐにわかります。ただし、Cobraには、よりアクセスしやすくするツールもあります。

## 演習 4：作成したテナントを GUI で表示する

Cobraの使用を容易にするツールについて説明する前に、APIC GUIで新しく作成したテナントを表示します。

1. Tenants [テナント]をクリックします。
2. ALL TENANTS [すべてのテナント]をクリックします。

Name	Alias	Description	Bridge Domains	VRFs	EPGs	Health Score
common			1	2	0	100
Heroes			1	1	5	94
infra			2	2	2	100
js_Cobra_Tenant			0	0	0	100
js_Toolkit_Tenant			0	0	0	100
mgmt			1	2	0	100
SnV			1	1	8	94

図1-37. 新しいテナント

### テナントを削除する

これらのテナントは、ACI ToolkitおよびCobraライブラリを紹介するためにのみ作成されました。先に進んで、それらを削除してください。

1. **自分の名前\_Cobra\_Tenant**を右クリックします。
2. Delete [削除]を選択します。

Name	Alias	Description	Bridge Domains	VRFs	EPGs	Health Score
common			1	2	0	100
Heroes			1	1	5	94
infra			2	2	2	100
js_Cobra_Tenant		Create Application Profile	0	0	0	100
js_Toolkit_Tenant		Create Bridge Domain	0	0	0	100
mgmt		Create VRF	1	2	0	100
SnV		Delete	1	1	8	94

図1-38. コブラテナントを削除

1. **自分の名前\_Toolkit\_Tenant**を右クリックします。
2. Delete [削除]を選択します。

The screenshot shows the Cisco Application Policy Infrastructure Controller (APIC) web interface. The top navigation bar includes links for System, Tenants (which is selected), Fabric, Virtual Networking, L4-L7 Services, Admin, Operations, Apps, and Integrations. The main content area displays a table titled 'All Tenants' with columns: Name, Alias, Description, Bridge Domains, VRFs, EPGs, and Health Score. The table lists several tenants: common, Heroes, Infra, js\_Toolkit\_Tenant, mgmt, and SnV. The 'js\_Toolkit\_Tenant' row is currently selected. A context menu is open over this row, with the 'Delete' option highlighted and surrounded by a red box. Other options in the menu include 'Create Application Profile', 'Create Bridge Domain', 'Create VRF', 'Save as ...', 'Post ...', 'Share', and 'Open In Object Store Browser'. The 'Delete' button is the final item in the list.

図1-39. ツールキットテナントを削除

## まとめ:ACIプログラマビリティエイド

CobraライブラリはToolkitライブラリを使用するよりも複雑であると以前に述べました。幸い、シスコは管理者を支援するいくつかのツールを提供しています。

お疲れ様でした！

次のラボでは、ACIツールキットについてより深いレベルで説明します。

### 3. ACI プログラム応用

#### 簡単な方法でACIスクリプトを作成する-WebArya

Aryaツールを活用して、APICをプログラムするPythonスクリプトをすばやく作成

#### ACI APIのピールバック

ACI RESTAPIとそれを簡単に使用するために利用できるツールを確認

#### Cobraを使用してACI Pythonスクリプトに「Bite」を追加する

Cobra SDKを使用してACIを管理するための強力なPythonスクリプトを作成することができます。

#### ACI Websockets

WebSocketを使用してアプリケーションをACIにリンクする方法を学びます。

#### 課題：Websocketを使用してWebexに通知を送信

コードへのあなたの番！Cobraを使用してACIヘルスダッシュボードを作成します。

#### 課題：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

Webexに送信されたACIネットワークの変更に関する即時通知を受け取ります

### 3.1. 簡単な方法で ACI スクリプトを作成する-WebArya

#### 目的

完了時間：60分

- Web Aryaを開始する
- APIC GUIからのJSON構成の取得
- JSONを使用してWeb AryaでサンプルPythonコードを作成する
- サンプルコードを更新して実行する

#### コードサンプル

- このラボのサンプルは、sbx-intermediate-aciにあります。

#### Arya-Cobraの複雑さを単純化する

(A)PIC (R)est p(Y)thon (A)dapterであるAryaは、Cobraライブラリを使用したPythonスクリプトの生成を支援します。Aryaは、XMLまたはJSON構成データのいずれかを入力として受け取り、同じ構成を生成するために使用できるPythonコードを出力します。(はい、Aryaは実際にコードを自動生成します！)

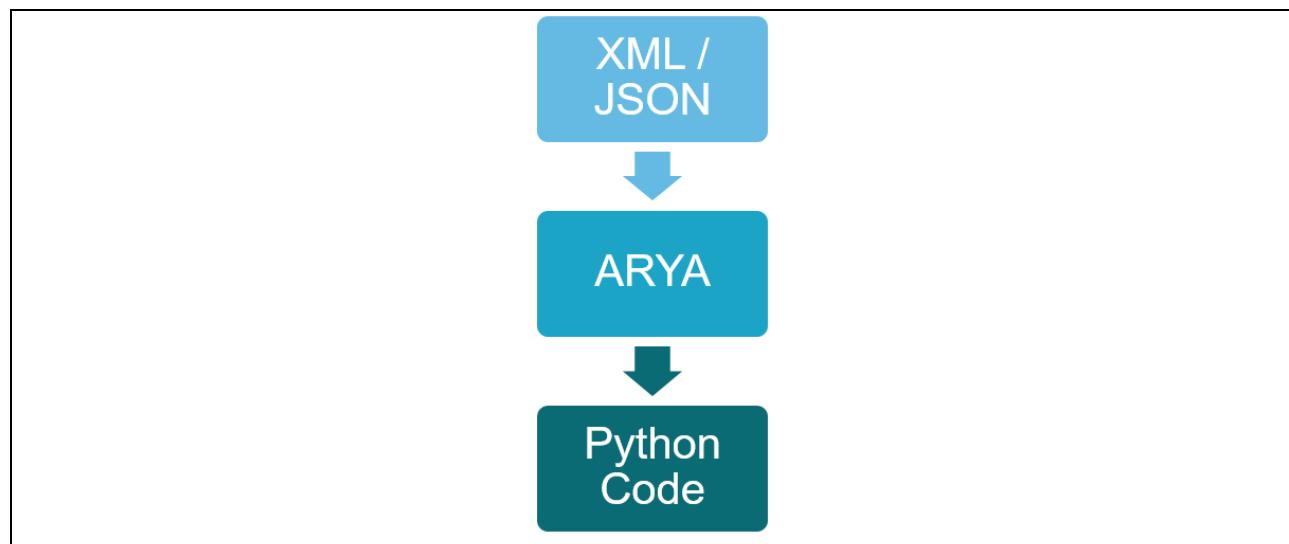


図1-40. Arya

スクリプトの正確性を常に確認し、必要な変更を加えてください。APICのURLと、スクリプトがAPICとの接続を確立するために使用するユーザ名とパスワードを変更する必要がある可能性があります。また、既存の構成のXMLまたはJSONを送信した場合は、新しいオブジェクトに必要な構成パラメータを変更する必要があります。

WebaryaラボではAryaの詳細が提供されており、Aryaを使用してCobraスクリプトを作成します。

#### 構成ソース

AryaがPythonスクリプトを自動的に生成する簡単な方法を提供することは素晴らしいですが、それでもAryaにロードするソース構成データを取得するという問題が残ります。APICには、構成データの提供を支援する3つのツールが付属しています。

- **GUIのダウンロード**機能を使用すると、XMLやJSONなどの既存のオブジェクトの設定をダウンロードすることができます。
- **Visore**は、XMLダウンロードオプションも備えたオブジェクトブラウザです。
- **APIインスペクタ**は、APIリクエストがAPICに行われると、URLおよびコンフィギュレーションデータを取得するために使用されます。

#### ステップ

1. WindowsマシンでPowerShellを開きます。
2. 次のコードリポジトリにあるベースラインスクリプトを実行します。

```
aci-learning-labs-code-samples/apic_fabric_setup/baseline.py
```

```
PS> cd aci-learning-labs-code-samples/apic_fabric_setup
```

```
PS> python baseline.py
```

# 期待される出力

```
Baselining APIC Simulator for Learning Labs
Setting up Fabric Nodes
```

```
Configuring Fabric Policies  
Setting up Common Tenant  
Setting up Heroes Tenant  
Setting up SnV Tenant
```

## ステップ 1: Arya とは何ですか？

AryaとPythonスクリプトの構築におけるその有用性についてはすでに簡単に紹介しました。Arya、または(APIC(R)ESTからp(Y)thon(A)dapterは、Cobraのクラスと関数を使用して新しい構成を構築するファイルを自動的に生成することにより、Pythonスクリプトの構築を容易にします。Aryaを使用する主な利点は次の3つです。

- 構成スクリプトの作成にかかる時間を短縮します
- APIドキュメントを読むよりも簡単
- 例によってAPIの使用方法を教えます

### Webarya

このラボでは、WebAryaを使用してPythonスクリプトを作成する方法について説明します。これは、Aryaと同じですが、使いやすいWebインターフェイスを提供します。「プログラマビリティオプション」ラボで、APICが構成データを収集するためのいくつかのオプションを提供していることを簡単に説明しました。この演習では、**GUIダウンロード**方法の使用について説明します。

## 演習 1：WebArya をインストール

この演習を続行するには、仮想環境にWeb Aryaをインストールする必要があります。Webaryaは<https://github.com/datacenter/webarya>にあります。

1. webaryaディレクトリに移動し、`pip install -r requirements.txt -U`を入力して、必要なPythonモジュールをインストールします。  

```
> cd aci-learning-labs-code-samples  
> cd sbx-intermediate-aci/sbx-intermediate-aci-00_webarya/  
> git clone https://github.com/datacenter/webarya.git  
> cd webarya  
> pip install -r requirements.txt -U
```

### WebAryaの使用

WebAryaは、ターミナルから起動されるFlaskベースのアプリケーションであり、任意のWebブラウザーで使用できます。アプリケーションは<http://0.0.0.0:8888>で実行されるため、複数のユーザが使用する中央サーバーをセットアップすることも、ローカルマシンで実行することもできます。

### WebAryaサービスの開始

Aryaがインストールされると、プログラムはwebarya/webaryaディレクトリに配置されます。ターミナルを開き、そのディレクトリを参照します。以下のディレクトリとファイルは、正しいディレクトリの下にリストされています。

```
> cd webarya  
> ls  
__init__.py README.md static templates webarya.py
```

webarya.pyプログラムを実行してWebAryaを起動します。WebAryaはデフォルトでポート8888を使用しますが、必要に応じて-pフラグを使用して、別のポートを使用できます。

```
> python webarya.py  
* Serving Flask app "webarya" (lazy loading)  
* Environment : production  
WARNING: This is a development server. Do not use in a production deployment.  
use a production WSGI server instead.  
* Debug mode: on  
* Restarting with stat  
* Debugger is active!  
* Debugger pin code: 102-464-421  
* Running on http://198.18.133.36:8888/ (Press CTRL+C to quit)
```

ターミナルを離れ、ブラウザを開いて<http://localhost:8888>にアクセスします。

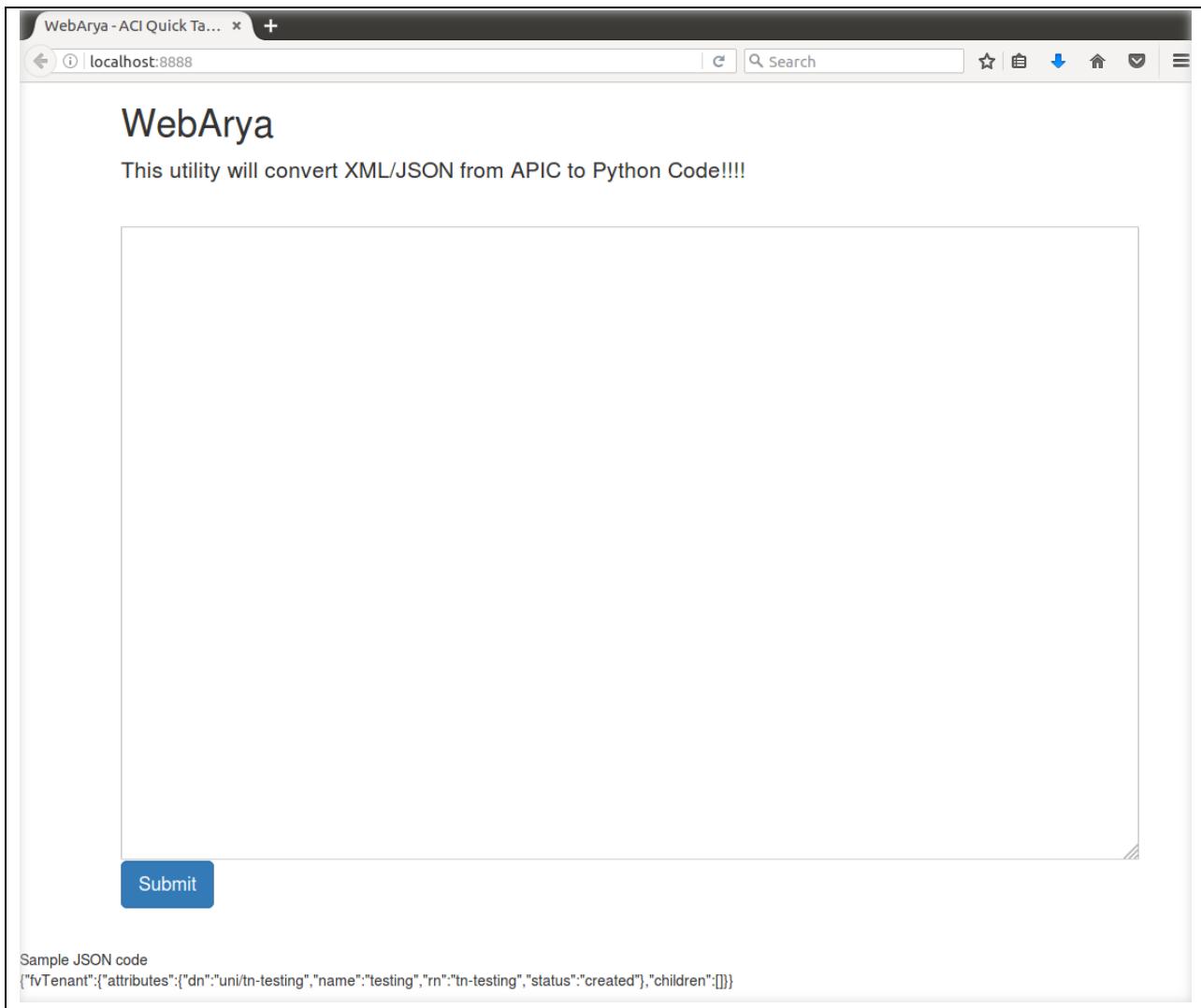


図1-41. WebAryaがスタートしました

ページの中央にある大きなボックスは、XMLまたはJSONが入力される場所であり、Submit [送信]をクリックすると、ターミナルプロンプトでAryaを使用した場合と同じ出力が返されます。

## 演習 2 : GUI で JSON をダウンロードする

WebAryaを使用するには、次の4つの基本的な手順があります。

1. GUIからサンプル構成データを収集する
2. サンプルデータをWebAryaへの入力として使用して、スクリプトを作成します
3. WebAryaの出力に必要な編集を加える
4. スクリプトを実行する

WebAryaを使用してスクリプトを作成する最初のステップは、入力ソースとして使用するサンプル構成ファイルを取得することです。このファイルを取得する一つの方法は、APIC GUIで既存の構成をダウンロードすることです。この演習の目標は、Herosテナント用の新しいアプリケーションを作成することです。シード構成には、既存の「Save\_The\_Planet」アプリケーションを使用できます。APIC GUIで次の手順に従います。

1. 「Heroes」という名前のテナントがAPIC GUIで構成されています。ウォークスルーを開始するには、APICを参照してログインします。

APIC Login Information	
URL	<a href="https://apic1.dcloud.cisco.com">https://apic1.dcloud.cisco.com</a>
User	admin

<b>Password</b>	C1sco12345
-----------------	------------

2. トップメニューのTenants[テナント]をクリックして、Herosテナントをダブルクリックで選択します。
  - 左側のメニューバーには、Herosテナントに関するオブジェクトが表示されます。
3. Application Profilesの横にある矢印をクリックして、子オブジェクトを開きます
  - アプリケーションSave\_The\_PlanetはApplication Profilesフォルダの下にあります
4. Save\_The\_Planetアプリケーションを右クリックして、Save as...を選択します。

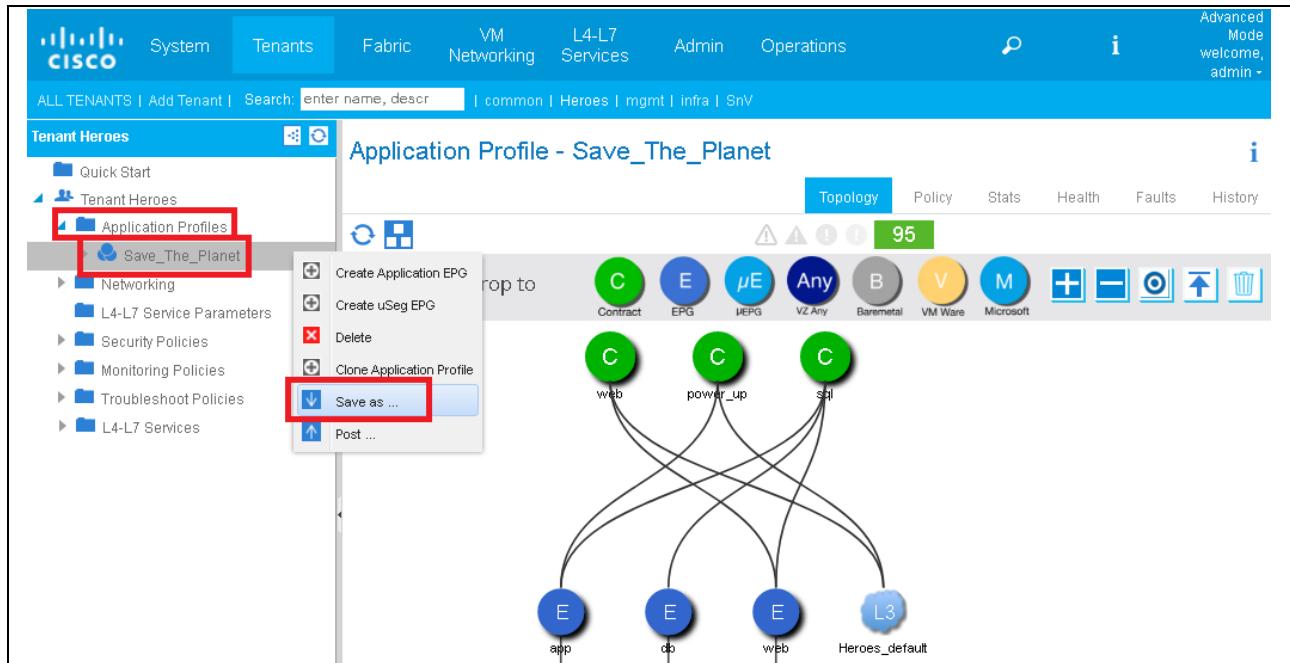


図1-42. WebAriaソース

ダウンロードオプションが表示された新しいボックスが表示されます

1. 次のオプションを選択します。
  - Only Configuration[構成のみ]
  - Subtree[サブツリー]
  - JSON
2. Download[ダウンロード]をクリックします

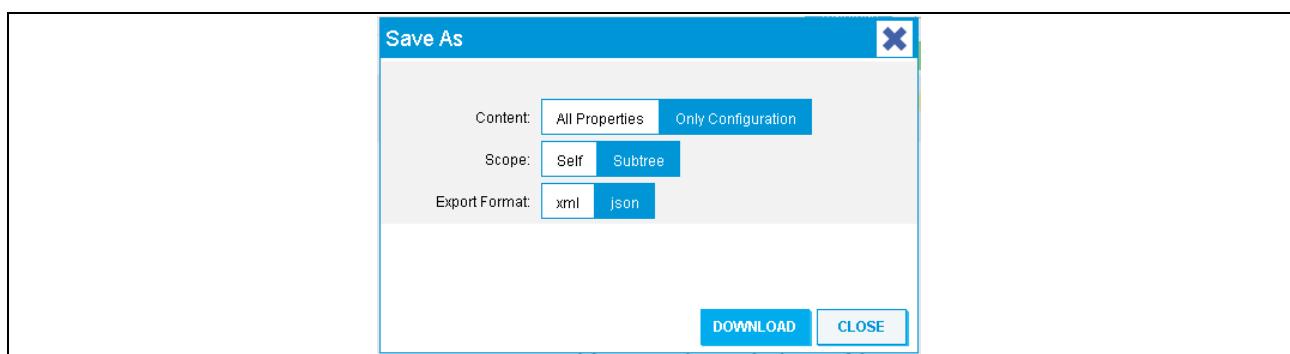


図1-43. WebAriaソースポップアップ

返されるJSONファイルがオブジェクトの構成プロパティのみを含むように制限されるため、Only Configuration[構成のみ]を選択しました。これは、WebAriaに送信するために必要なすべてです。統計やカウンターなどの非構成プロパティは、後で振り返る必要がある場合に読みにくくする可能性があります。

「サブツリー」オプションには、管理対象オブジェクトのすべての子が含まれます。つまり、WebAriaは、完全なアプリケーションプロファイルを構成するより堅牢なスクリプトを提供します。

WebAriaはXMLまたはJSONをサポートしているため、この演習ではJSONを使用します。

名前の付いたファイルap-Save\_The\_Planet.jsonがブラウザのダウンロードディレクトリにダウンロードされます。

## 演習 3 : Cobra コードを収集する

いくつかの構成データができたので、WebAryaを使用してCobraスクリプトを取得する準備が整いました。

### サンプル構成をコピー＆ペーストする

ダウンロードしたファイルを開き、最初のステップで開いたWebAryaのメインページにすべてのコンテンツをコピーします。

```
{"totalCount": "1", "imdata": [{"fvAp": {"attributes": {"descr": "", "dn": "uni/tn-Heroes/ap-Save_The_Planet", "name": "Save_The_Planet", "ownerKey": "", "ownerTag": "", "prio": "unspecified"}, "children": [{"fvAEPg": {"attributes": {"descr": "", "isAttrBasedEPg": "no", "matchT": "AtleastOne", "name": "app", "prio": "unspecified"}, "children": [{"fvRsCons": {"attributes": {"prio": "unspecified", "tnVzBrCPName": "sql"}}, {"fvRsPathAtt": {"attributes": {"desc": "vlan-201", "instrImedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2B]"}}, {"fvRsPathAtt": {"attributes": {"desc": "vlan-201", "instrImedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2A]"}}, {"fvRsDomAtt": {"attributes": {"encap": "unknown", "instrImedcy": "lazy", "resImedcy": "lazy", "tDn": "uni/phys-Heroes_phys"}}, {"fvRsCustQosPol": {"attributes": {"tnQosCustomPolName": ""}}, {"fvRsBd": {"attributes": {"tnFvBDName": "Hero_Land"}}, {"fvRsProv": {"attributes": {"matchT": "AtleastOne", "prio": "unspecified", "tnVzBrCPName": "power_up"}}, {"fvAEPg": {"attributes": {"desc": "", "isAttrBasedEPg": "no", "matchT": "AtleastOne", "name": "db", "prio": "unspecified"}, "children": [{"fvRsPathAtt": {"attributes": {"desc": "", "encap": "vlan-202", "instrImedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2B]"}}, {"fvRsPathAtt": {"attributes": {"desc": "", "encap": "vlan-202", "instrImedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2A]"}}, {"fvRsDomAtt": {"attributes": {"encap": "unknown", "instrImedcy": "lazy", "resImedcy": "lazy", "tDn": "uni/phys-Heroes_phys"}}, {"fvRsCustQosPol": {"attributes": {"tnQosCustomPolName": ""}}, {"fvRsBd": {"attributes": {"tnFvBDName": "Hero_Land"}}, {"fvRsProv": {"attributes": {"matchT": "AtleastOne", "prio": "unspecified", "tnVzBrCPName": "sql"}}, {"fvAEPg": {"attributes": {"desc": "", "isAttrBasedEPg": "no", "matchT": "AtleastOne", "name": "web", "prio": "unspecified"}, "children": [{"fvRsCons": {"attributes": {"prio": "unspecified", "tnVzBrCPName": "sql"}}, {"fvRsPathAtt": {"attributes": {"desc": "", "encap": "vlan-200", "instrImedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2B]"}}, {"fvRsPathAtt": {"attributes": {"desc": "", "encap": "vlan-200", "instrImedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2A]"}}, {"fvRsDomAtt": {"attributes": {"encap": "unknown", "instrImedcy": "lazy", "resImedcy": "lazy", "tDn": "uni/phys-Heroes_phys"}}, {"fvRsCustQosPol": {"attributes": {"tnQosCustomPolName": ""}}, {"fvRsBd": {"attributes": {"tnFvBDName": "Hero_Land"}}, {"fvRsProv": {"attributes": {"matchT": "AtleastOne", "prio": "unspecified", "tnVzBrCPName": "web"}}}]}]}]}]}]}]}
```

この構成ファイルは、サンプルコードリポジトリのようにap-Save\_The\_Planet-config.jsonとして保存されています。

WebArya - ACI Quick Ta... +

localhost:8888

Search

# WebArya

This utility will convert XML/JSON from APIC to Python Code!!!!

```
{"totalCount": "1", "imdata": [{"fvAp": {"attributes": {"descr": "", "dn": "uni/tn-Heroes/ap-Save_The_Planet"}, "name": "Save_The_Planet", "ownerKey": "", "ownerTag": "", "prio": "unspecified"}, "children": [{"fvAEPg": {"attributes": {"descr": "", "isAttrBasedEPg": "no", "matchT": "AtleastOne", "name": "app", "prio": "unspecified"}, "children": [{"fvRsCons": {"attributes": {"prio": "unspecified", "tnVzBrCPName": "sql"}}, {"fvRsPathAtt": {"attributes": {"descr": "", "encap": "vlan-201", "instlMedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2B]"}}, {"fvRsPathAtt": {"attributes": {"descr": "", "encap": "vlan-201", "instlMedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2A]"}}, {"fvRsDomAtt": {"attributes": {"encap": "unknown", "instlMedcy": "lazy", "reslMedcy": "lazy", "tDn": "uni/phys-Heroes_phys"}}, {"fvRsCustQosPol": {"attributes": {"inQosCustomPolName": ""}}}, {"fvRsBd": {"attributes": {"inFvBDName": "Hero_Land"}}, {"fvRsProv": {"attributes": {"matchT": "AtleastOne", "prio": "unspecified", "inVzBrCPName": "power_up"}}}]}], "fvAEPg": {"attributes": {"descr": "", "isAttrBasedEPg": "no", "matchT": "AtleastOne", "name": "db", "prio": "unspecified"}, "children": [{"fvRsPathAtt": {"attributes": {"descr": "", "encap": "vlan-202", "instlMedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2B]"}}, {"fvRsPathAtt": {"attributes": {"descr": "", "encap": "vlan-202", "instlMedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2A]"}}, {"fvRsDomAtt": {"attributes": {"encap": "unknown", "instlMedcy": "lazy", "reslMedcy": "lazy", "tDn": "uni/phys-Heroes_phys"}}, {"fvRsCustQosPol": {"attributes": {"inQosCustomPolName": ""}}}, {"fvRsBd": {"attributes": {"inFvBDName": "Hero_Land"}}, {"fvRsProv": {"attributes": {"matchT": "AtleastOne", "prio": "unspecified", "inVzBrCPName": "sql"}}}]}], "fvAEPg": {"attributes": {"descr": "", "isAttrBasedEPg": "no", "matchT": "AtleastOne", "name": "web", "prio": "unspecified"}, "children": [{"fvRsCons": {"attributes": {"prio": "unspecified", "tnVzBrCPName": "sql"}}, {"fvRsPathAtt": {"attributes": {"descr": "", "encap": "vlan-200", "instlMedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2B]"}}, {"fvRsPathAtt": {"attributes": {"descr": "", "encap": "vlan-200", "instlMedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2A]"}}, {"fvRsDomAtt": {"attributes": {"encap": "unknown", "instlMedcy": "lazy", "reslMedcy": "lazy", "tDn": "uni/phys-Heroes_phys"}}, {"fvRsCustQosPol": {"attributes": {"inQosCustomPolName": ""}}}, {"fvRsBd": {"attributes": {"inFvBDName": "Hero_Land"}}, {"fvRsProv": {"attributes": {"matchT": "AtleastOne", "prio": "unspecified", "inVzBrCPName": "web"}}}]}]}]}]
```

Submit

Sample JSON code

```
{"fvTenant": {"attributes": {"dn": "uni/tn-testing", "name": "testing", "rn": "tn-testing", "status": "created"}, "children": []}}
```

図1-44. WebAryaコピー構成

Web Aryaページ下部のSubmitをクリックして、Cobraの出力を返します。

```

Success - ACI Quick Tasks
localhost:8888
#!/usr/bin/env python
"""

Autogenerated code using webarya.py
Original Object Document Input:
{"totalCount": "1", "imdata": [{"fvAp": {"attributes": {"descr": "", "dn": "uni/tn-Heroes_ap-Save_The_Planet", "name": "Save_The_Planet", "ownerKey": "", "ownerTag": "", "prio": "unspecified"}, "children": [{"fvAEPg": {"attributes": {"descr": "", "isAttrBasedEPg": "no", "matchT": "AtleastOne", "name": "app", "prio": "unspecified"}, "children": [{"fvRsCons": {"attributes": {"prio": "unspecified", "tnVzBrCPName": "sql"}}, {"fvRsPathAtt": {"attributes": {"descr": "", "encap": "vlan-201", "instrImedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2B]"}}, {"fvRsPathAtt": {"attributes": {"descr": "", "encap": "vlan-201", "instrImedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2A]"}}, {"fvRsDomAtt": {"attributes": {"encap": "unknown", "instrImedcy": "lazy", "resMedcy": "lazy", "tDn": "uni/phys-Heroes_phys"}}, {"fvRsCustQosPol": {"attributes": {"InQosCustomPolName": ""}}, {"fvRsBd": {"attributes": {"tnFvBDName": "Hero_Land"}}}}, {"fvRsProv": {"attributes": {"matchT": "AtleastOne", "prio": "unspecified", "tnVzBrCPName": "power_up"}}}}], {"fvAEPg": {"attributes": {"descr": "", "isAttrBasedEPg": "no", "matchT": "AtleastOne", "name": "db", "prio": "unspecified"}, "children": [{"fvRsPathAtt": {"attributes": {"descr": "", "encap": "vlan-202", "instrImedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2B]"}}, {"fvRsPathAtt": {"attributes": {"descr": "", "encap": "vlan-202", "instrImedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2A]"}}, {"fvRsDomAtt": {"attributes": {"encap": "unknown", "instrImedcy": "lazy", "resMedcy": "lazy", "tDn": "uni/phys-Heroes_phys"}}, {"fvRsCustQosPol": {"attributes": {"InQosCustomPolName": ""}}, {"fvRsBd": {"attributes": {"tnFvBDName": "Hero_Land"}}}}, {"fvRsProv": {"attributes": {"matchT": "AtleastOne", "prio": "unspecified", "tnVzBrCPName": "sql"}}}], {"fvAEPg": {"attributes": {"descr": "", "isAttrBasedEPg": "no", "matchT": "AtleastOne", "name": "web", "prio": "unspecified"}, "children": [{"fvRsCons": {"attributes": {"prio": "unspecified", "tnVzBrCPName": "sql"}}, {"fvRsPathAtt": {"attributes": {"descr": "", "encap": "vlan-200", "instrImedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2B]"}}, {"fvRsPathAtt": {"attributes": {"descr": "", "encap": "vlan-200", "instrImedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2A]"}}, {"fvRsDomAtt": {"attributes": {"encap": "unknown", "instrImedcy": "lazy", "resMedcy": "lazy", "tDn": "uni/phys-Heroes_phys"}}, {"fvRsCustQosPol": {"attributes": {"InQosCustomPolName": ""}}, {"fvRsBd": {"attributes": {"tnFvBDName": "Hero_Land"}}}}, {"fvRsProv": {"attributes": {"matchT": "AtleastOne", "prio": "unspecified", "tnVzBrCPName": "web"}}}]}]}]}}

raise RuntimeError('Please review the auto generated code before ' +
    'executing the output. Some placeholders will ' +
    'need to be changed')

# list of packages that should be imported for this code to work
import cobra.mit.access
import cobra.mit.request
import cobra.mit.session
import cobra.model.fv
import cobra.model.pol
from cobra.internal.codec.xmlcodec import toXMLStr

# log into an APIC and create a directory object
ls = cobra.mit.session.LoginSession('https://1.1.1.1', 'admin', 'password')
md = cobra.mit.access.MoDirectory(ls)
md.login()

# the top level object on which operations will be made
polUni = cobra.model.pol.Uni('')
fvTenant = cobra.model.fv.Tenant(polUni, 'Heroes')

# build the request using cobra syntax

```

図1-45. WebAryaサンプルスクリプト

### Pythonコードをキャプチャする

次に、結果のコードをテキストエディタまたはIDEで開きます。ブラウザでCtrl-Aを選択してWebArya出力の内容を全てコピーし、新しい空白のドキュメントに貼り付けます。Pythonスクリプトには注意が必要な場所がいくつかあるため、作業ディレクトリ(aci-learning-labs-code-samples\$sbx-intermediate-aci\$sbx-intermediate-aci-00\_webarya)にadd\_application.pyとして保存します。

### 結果のスクリプトの内容

```

#!/usr/bin/env python
"""

Autogenerated code using webarya.py
Original Object Document Input:
{"totalCount": "1", "imdata": [{"fvAp": {"attributes": {"descr": "", "dn": "uni/tn-Heroes_ap-Save_The_Planet", "name": "Save_The_Planet", "ownerKey": "", "ownerTag": "", "prio": "unspecified"}, "children": [{"fvAEPg": {"attributes": {"descr": "", "isAttrBasedEPg": "no", "matchT": "AtleastOne", "name": "app", "prio": "unspecified"}, "children": [{"fvRsCons": {"attributes": {"prio": "unspecified", "tnVzBrCPName": "sql"}}, {"fvRsPathAtt": {"attributes": {"descr": "", "encap": "vlan-201", "instrImedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2B]"}}, {"fvRsPathAtt": {"attributes": {"descr": "", "encap": "vlan-201", "instrImedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2A]"}}, {"fvRsDomAtt": {"attributes": {"encap": "unknown", "instrImedcy": "lazy", "resMedcy": "lazy", "tDn": "uni/phys-Heroes_phys"}}, {"fvRsCustQosPol": {"attributes": {"InQosCustomPolName": ""}}, {"fvRsBd": {"attributes": {"tnFvBDName": "Hero_Land"}}}}, {"fvRsProv": {"attributes": {"matchT": "AtleastOne", "prio": "unspecified", "tnVzBrCPName": "power_up"}}}}], {"fvAEPg": {"attributes": {"descr": "", "isAttrBasedEPg": "no", "matchT": "AtleastOne", "name": "db", "prio": "unspecified"}, "children": [{"fvRsPathAtt": {"attributes": {"descr": "", "encap": "vlan-202", "instrImedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2B]"}}, {"fvRsPathAtt": {"attributes": {"descr": "", "encap": "vlan-202", "instrImedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2A]"}}, {"fvRsDomAtt": {"attributes": {"encap": "unknown", "instrImedcy": "lazy", "resMedcy": "lazy", "tDn": "uni/phys-Heroes_phys"}}, {"fvRsCustQosPol": {"attributes": {"InQosCustomPolName": ""}}, {"fvRsBd": {"attributes": {"tnFvBDName": "Hero_Land"}}}}, {"fvRsProv": {"attributes": {"matchT": "AtleastOne", "prio": "unspecified", "tnVzBrCPName": "sql"}}}], {"fvAEPg": {"attributes": {"descr": "", "isAttrBasedEPg": "no", "matchT": "AtleastOne", "name": "web", "prio": "unspecified"}, "children": [{"fvRsCons": {"attributes": {"prio": "unspecified", "tnVzBrCPName": "sql"}}, {"fvRsPathAtt": {"attributes": {"descr": "", "encap": "vlan-200", "instrImedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2B]"}}, {"fvRsPathAtt": {"attributes": {"descr": "", "encap": "vlan-200", "instrImedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2A]"}}, {"fvRsDomAtt": {"attributes": {"encap": "unknown", "instrImedcy": "lazy", "resMedcy": "lazy", "tDn": "uni/phys-Heroes_phys"}}, {"fvRsCustQosPol": {"attributes": {"InQosCustomPolName": ""}}, {"fvRsBd": {"attributes": {"tnFvBDName": "Hero_Land"}}}}, {"fvRsProv": {"attributes": {"matchT": "AtleastOne", "prio": "unspecified", "tnVzBrCPName": "web"}}}]}]}]}}

raise RuntimeError('Please review the auto generated code before ' +
    'executing the output. Some placeholders will ' +
    'need to be changed')

# list of packages that should be imported for this code to work
import cobra.mit.access
import cobra.mit.request
import cobra.mit.session
import cobra.model.fv
import cobra.model.pol
from cobra.internal.codec.xmlcodec import toXMLStr

# log into an APIC and create a directory object
ls = cobra.mit.session.LoginSession('https://1.1.1.1', 'admin', 'password')
md = cobra.mit.access.MoDirectory(ls)
md.login()

# the top level object on which operations will be made
polUni = cobra.model.pol.Uni('')
fvTenant = cobra.model.fv.Tenant(polUni, 'Heroes')

# build the request using cobra syntax

```

```

protpaths-101-102/pathep-[Heroes_FI-2B"]}}}, {"fvRsPathAtt": {"attributes": {"descr": "", "encap": "vlan-202", "instrImedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2A]"}}, {"fvRsDomAtt": {"attributes": {"encap": "unknown", "instrImedcy": "lazy", "resImedcy": "lazy", "tDn": "uni/phys-Heroes_phys"}}, {"fvRsCustQosPol": {"attributes": {"tnQosCustomPolName": ""}}}, {"fvRsBd": {"attributes": {"tnFvBDName": "Hero_Land"}}, {"fvRsProv": {"attributes": {"matchT": "AtleastOne", "prio": "unspecified", "tnVzBrCPName": "sql"}}, {"fvAEPg": {"attributes": {"descr": "", "isAttrBasedEPg": "no", "matchT": "AtleastOne", "name": "web", "prio": "unspecified"}, "children": [{"fvRsCons": {"attributes": {"prio": "unspecified", "tnVzBrCPName": "sql"}}, {"fvRsPathAtt": {"attributes": {"descr": "", "encap": "vlan-200", "instrImedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2B]"}}, {"fvRsPathAtt": {"attributes": {"descr": "", "encap": "vlan-200", "instrImedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2A]"}}, {"fvRsDomAtt": {"attributes": {"encap": "unknown", "instrImedcy": "lazy", "resImedcy": "lazy", "tDn": "uni/phys-Heroes_phys"}}, {"fvRsCustQosPol": {"attributes": {"tnQosCustomPolName": ""}}}, {"fvRsBd": {"attributes": {"tnFvBDName": "Hero_Land"}}, {"fvRsProv": {"attributes": {"matchT": "AtleastOne", "prio": "unspecified", "tnVzBrCPName": "web"}}}]}]}}

'''  

raise RuntimeError('Please review the auto generated code before ' +  

'executing the output. Some placeholders will ' +  

'need to be changed')

# list of packages that should be imported for this code to work
import cobra.mit.access
import cobra.mit.request
import cobra.mit.session
import cobra.model.fv
import cobra.model.pol
from cobra.internal.codec.xmlcodec import toXMLStr

# log into an APIC and create a directory object
ls = cobra.mit.session.LoginSession('https://1.1.1.1', 'admin', 'password')
md = cobra.mit.access.MoDirectory(ls)
md.login()

# the top level object on which operations will be made
polUni = cobra.model.pol.Uni('')
fvTenant = cobra.model.fv.Tenant(polUni, 'Heroes')

# build the request using cobra syntax
fvAp = cobra.model.fv.Ap(fvTenant, ownerKey=u'', name=u'Save_The_Planet', prio=u'unspecified', ownerTag=u'', descr=u'')
fvAEPg = cobra.model.fv.AEPg(fvAp, isAttrBasedEPg=u'no', matchT=u'AtleastOne', prio=u'unspecified', name=u'app', descr=u'')
fvRsCons = cobra.model.fv.RsCons(fvAEPg, tnVzBrCPName=u'sql', prio=u'unspecified')
fvRsPathAtt = cobra.model.fv.RsPathAtt(fvAEPg,
tDn=u'topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2B]', instrImedcy=u'lazy', encap=u'vlan-201', descr=u'', mode=u'regular')
fvRsPathAtt2 = cobra.model.fv.RsPathAtt(fvAEPg,
tDn=u'topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2A]', instrImedcy=u'lazy', encap=u'vlan-201', descr=u'', mode=u'regular')
fvRsDomAtt = cobra.model.fv.RsDomAtt(fvAEPg, instrImedcy=u'lazy', resImedcy=u'lazy', encap=u'unknown', tDn=u'uni/phys-Heroes_phys')
fvRsCustQosPol = cobra.model.fv.RsCustQosPol(fvAEPg, tnQosCustomPolName=u'')
fvRsBd = cobra.model.fv.RsBd(fvAEPg, tnFvBDName=u'Hero_Land')
fvRsProv = cobra.model.fv.RsProv(fvAEPg, tnVzBrCPName=u'power_up', matchT=u'AtleastOne', prio=u'unspecified')
fvAEPg2 = cobra.model.fv.AEPg(fvAp, isAttrBasedEPg=u'no', matchT=u'AtleastOne', prio=u'unspecified', name=u'db', descr=u'')
fvRsPathAtt3 = cobra.model.fv.RsPathAtt(fvAEPg2,
tDn=u'topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2B]', instrImedcy=u'lazy', encap=u'vlan-202', descr=u'', mode=u'regular')
fvRsPathAtt4 = cobra.model.fv.RsPathAtt(fvAEPg2,
tDn=u'topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2A]', instrImedcy=u'lazy', encap=u'vlan-202', descr=u'', mode=u'regular')
fvRsDomAtt2 = cobra.model.fv.RsDomAtt(fvAEPg2, instrImedcy=u'lazy', resImedcy=u'lazy', encap=u'unknown', tDn=u'uni/phys-Heroes_phys')

```

```

fvRsCustQosPol2 = cobra.model.fv.RsCustQosPol(fvAEPg2, tnQosCustomPolName=u'')
fvRsBd2 = cobra.model.fv.RsBd(fvAEPg2, tnFvBDName=u'Hero_Land')
fvRsProv2 = cobra.model.fv.RsProv(fvAEPg2, tnVzBrCPName=u'sql', matchT=u'AtleastOne',
prio=u'unspecified')
fvAEPg3 = cobra.model.fv.AEPg(fvAp, isAttrBasedEPg=u'no', matchT=u'AtleastOne',
prio=u'unspecified', name=u'web', descr=u'')
fvRsCons2 = cobra.model.fv.RsCons(fvAEPg3, tnVzBrCPName=u'sql', prio=u'unspecified')
fvRsPathAtt5 = cobra.model.fv.RsPathAtt(fvAEPg3,
tDn=u'topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2B]', instrImedcy=u'lazy',
encap=u'vlan-200', descr=u'', mode=u'regular')
fvRsPathAtt6 = cobra.model.fv.RsPathAtt(fvAEPg3,
tDn=u'topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2A]', instrImedcy=u'lazy',
encap=u'vlan-200', descr=u'', mode=u'regular')
fvRsDomAtt3 = cobra.model.fv.RsDomAtt(fvAEPg3, instrImedcy=u'lazy',
resImedcy=u'lazy', encap=u'unknown', tDn=u'uni/phys-Heroes_phys')
fvRsCustQosPol3 = cobra.model.fv.RsCustQosPol(fvAEPg3, tnQosCustomPolName=u'')
fvRsBd3 = cobra.model.fv.RsBd(fvAEPg3, tnFvBDName=u'Hero_Land')
fvRsProv3 = cobra.model.fv.RsProv(fvAEPg3, tnVzBrCPName=u'web', matchT=u'AtleastOne',
prio=u'unspecified')

# commit the generated code to APIC
print toXMLStr(fvTenant)
c = cobra.mit.request.ConfigRequest()
c.addMo(fvTenant)
md.commit(c)

```

## 演習4：ターミナルとコードノート

この演習では、必ずPythonのコーディング端末を開き、このラボのコードサンプルのディレクトリを入力してください。

```
>
aci-learning-labs-code-samples/sbx-intermediate-aci/sbx-intermediate-aci-00_webar
ya
```

### 新しいPythonファイルを作成する

演習3で保存したファイルの各セクションをウォータースルーし、ファイルの用途と変更方法について説明します。

### シバン

スクリプトの最初の行はシバン行です。これは、ファイルの実行時に使用するPythonをシステムに通知します。これにより、ユーザは./program.pyの代わりにpython program.pyを入力できます。この行は変更しないでください。この実行方法が機能するには、ファイルに実行ビットが設定されている必要があります。

```
#!/usr/bin/env python
```

### 既存のドキュメントを保持する

次のセクションは、スクリプトの生成に使用されるJSONデータを提供する複数行コメントです。ドキュメント化のためにコメントをファイルに保存しておくことをお勧めします。そのため、サンプルデータをダウンロードするときにConfiguration Only[構成のみ]を選択することをお勧めします。これらの行はスクリプトに残してください。

```
'''
Autogenerated code using webarya.py
Original Object Document Input:
{
  "totalCount": "1",
  "imdata": [
    {
      "fvAp": {
        "attributes": {
          "descr": "",
          "dn": "uni/tn-Heroes/ap-Save_The_Planet",
          "name": "Save_The_Planet",
          "ownerKey": "",
          "ownerTag": "",
          "prio": "unspecified"
        },
        "children": [
          {
            "fvAEPg": {
              "attributes": {
                "descr": "",
                "isAttrBasedEPg": "no",
                "matchT": "AtleastOne",
                "name": "app",
                "prio": "unspecified"
              },
              "children": [
                {
                  "fvRsCons": {
                    "attributes": {
                      "prio": "unspecified",
                      "tnVzBrCPName": "sql"
                    }
                  },
                  {
                    "fvRsPathAtt": {
                      "attributes": {
                        "descr": "",
                        "encap": "vlan-201",
                        "instrImedcy": "lazy",
                        "mode": "regular",
                        "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2B]"
                      }
                    },
                    {
                      "fvRsPathAtt": {
                        "attributes": {
                          "descr": "",
                          "encap": "vlan-201",
                          "instrImedcy": "lazy",
                          "mode": "regular",
                          "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2A]"
                        }
                      },
                      {
                        "fvRsDomAtt": {
                          "attributes": {
                            "encap": "unknown",
                            "instrImedcy": "lazy",
                            "resImedcy": "lazy",
                            "tDn": "uni/phys-Heroes_phys"
                          }
                        },
                        {
                          "fvRsCustQosPol": {
                            "attributes": {
                              "tnQosCustomPolName": ""
                            }
                          },
                          {
                            "fvRsBd": {
                              "attributes": {
                                "tnFvBDName": "Hero_Land"
                              }
                            },
                            {
                              "fvRsProv": {
                                "attributes": {
                                  "matchT": "AtleastOne",
                                  "prio": "unspecified",
                                  "tnVzBrCPName": "power_up"
                                }
                              }
                            }
                          }
                        }
                      }
                    }
                  }
                }
              ]
            }
          }
        ]
      }
    }
  ]
}
```

```

vAEPg": {"attributes": {"descr": "", "isAttrBasedEPg": "no", "matchT": "AtleastOne", "name": "db", "prio": "unspecified"}, "children": [{"fvRsPathAtt": {"attributes": {"descr": "", "encap": "vlan-202", "instrImedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2B]"}}, {"fvRsPathAtt": {"attributes": {"descr": "", "encap": "vlan-202", "instrImedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2A]"}}, {"fvRsDomAtt": {"attributes": {"encap": "unknown", "instrImedcy": "lazy", "resImedcy": "lazy", "tDn": "uni/phys-Heroes_phys"}}, {"fvRsCustQosPol": {"attributes": {"tnQosCustomPolName": ""}}}, {"fvRsBd": {"attributes": {"tnFvBDName": "Hero_Land"}}, {"fvRsProv": {"attributes": {"matchT": "AtleastOne", "prio": "unspecified", "tnVzBrCPName": "sql"}}, {"fvAEPg": {"attributes": {"descr": "", "isAttrBasedEPg": "no", "matchT": "AtleastOne", "name": "web", "prio": "unspecified"}, "children": [{"fvRsCons": {"attributes": {"prio": "unspecified", "tnVzBrCPName": "sql"}}, {"fvRsPathAtt": {"attributes": {"descr": "", "encap": "vlan-200", "instrImedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2B]"}}, {"fvRsPathAtt": {"attributes": {"descr": "", "encap": "vlan-200", "instrImedcy": "lazy", "mode": "regular", "tDn": "topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2A]"}}, {"fvRsDomAtt": {"attributes": {"encap": "unknown", "instrImedcy": "lazy", "resImedcy": "lazy", "tDn": "uni/phys-Heroes_phys"}}, {"fvRsCustQosPol": {"attributes": {"tnQosCustomPolName": ""}}}, {"fvRsBd": {"attributes": {"tnFvBDName": "Hero_Land"}}, {"fvRsProv": {"attributes": {"matchT": "AtleastOne", "prio": "unspecified", "tnVzBrCPName": "web"}}}]}]}]}}
  
```

## エラーを発生させる

このセクションは、スクリプトの実行を妨げることを目的としたAryaによって提供される警告メッセージです。これは、コードにいくつかの変更を加える必要があります、実行前に確認する必要があることを思い出させてくれます。

これらの行をスクリプトから削除します。

```

raise RuntimeError('Please review the auto generated code before ' +
'executing the output. Some placeholders will ' +
'need to be changed')
  
```

## ライブラリのインポート

ファイルの残りの部分は、新しいオブジェクトを構成する実際のコードです。まず、必要なライブラリをインポートする必要があります。

```

# list of packages that should be imported for this code to work
import cobra.mit.access
import cobra.mit.request
import cobra.mit.session
import cobra.model.fv
import cobra.model.pol
from cobra.internal.codec.xmlcodec import toXMLStr
  
```

このラボでは認証にcredentials.pyファイルを使用しています。InsecureRequestWarning: Unverified HTTPS request is being made.エラー/警告を抑制するには、次のコードを追加します。

```

from credentials import *

import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)
  
```

## スクリプトに変数を追加する

以下の構成変数のセクションを追加します。テナント名にはイニシャルと電話番号を使用します。テナントが存在しない場合は、スクリプトの実行時に作成されます。

```

# configuration variables
tenant = 'jt_1234'
bridge_domain = 'Hero_Land'
application = 'Save_The_Network'
vlan1 = 'vlan-211'
vlan2 = 'vlan-212'
vlan3 = 'vlan-210'
  
```

## APICログイン

次のステップは、ログイン情報を提供し、APICとの接続を確立することです。URL、ユーザ名、およびパスワードを更新して、credentials.pyをインポートして変数を使用します。

```

import credential
# log into an APIC and create a directory object
  
```

```
ls = cobra.mit.session.LoginSession(URL, LOGIN, PASSWORD)
md = cobra.mit.access.MoDirectory(ls)
md.login()
```

## トップレベルのオブジェクトを作成する

コードのこのセクションは、必要な親オブジェクトを作成するためのものです。polUniはツリーのルートであり、fvTenantを使用してルートからテナントオブジェクトを識別します。

変数tenantを使用して、テナント名を[Heros]から変更します。

```
# the top level object on which operations will be made
polUni = cobra.model.pol.Uni('')
fvTenant = cobra.model.fv.Tenant(polUni, tenant)
```

## 新しい構成を作成する

次の主要なコードブロックは、Save\_The\_Planetアプリケーションの実際の構成です。これには、新しいSave\_The\_Networkアプリケーション用にいくつかのフィールドを更新する必要があります。これはすべて1つの大きなブロックですが、各サブセクションの目的について説明するために分割します。

各行の目的を説明するコメントがコードブロック内に追加されています。これらをファイルに追加する必要はありません。のコメント##は、次の行で更新が必要であることを示すために使用されます。

## アプリケーションとEPGを変更する

コードの最初の行は、EPGの親オブジェクトである新しいアプリケーションプロファイルを作成します。作成する3つのEPGは同じアプリケーションに属しているため、1つのアプリケーションオブジェクトを作成するだけで、新しいEPGはそれぞれこの管理対象オブジェクトを参照します。

コードの最初に作成された変数applicationを使用するようにname引数を変更します。

```
## Application名"Save_The_Planet" を application に変更します。
fvAp = cobra.model.fv.Ap(fvTenant, ownerKey=u'', name=application,
prio=u'unspecified', ownerTag=u'', descr=u'')
```

## 「アpri」EPGを作成する

コードの最初の数行は、すべて同じEPGに関連しているため、グループ化できます。EPGの名前は「Save\_The\_Planet」のEPGと同じになるため、新しいEPGオブジェクトを作成するための構成行を変更する必要はありません。また、EPGは、オブジェクトfvAPを参照することによってアプリケーションプロファイルに関連付けられます。そのオブジェクトが変更されており、これが前の手順で行われている限り、アリプロファイルオブジェクトは新しいアリプロファイルを参照します。

変数vlan1とbridge\_domainを使用するようにencap引数とtnFvBDName引数を変更します。

更新が必要な行に##は、生成されたコードではなく、以下のコードで始まるコメントが含まれます。

```
# create the first EPG underneath your new Application.
fvAEPg = cobra.model.fv.AEPg(fvAp, isAttrBasedEPg=u'no', matchT=u'AtleastOne',
prio=u'unspecified', name=u'app', descr=u'')

# have the EPG created by the previous line of code consume the "sql" contract.
fvRsCons = cobra.model.fv.RsCons(fvAEPg, tnVzBrCPName=u'sql', prio=u'unspecified')

# assign a VLAN and vPC to the EPG represented by fvAEPg
## encapをvlan1に変更
fvRsPathAtt = cobra.model.fv.RsPathAtt(fvAEPg,
tDn=u'topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2B]', instrImedcy=u'lazy',
encap=vlan1, descr=u'', mode=u'regular')

# assign a VLAN and a second vPC to the EPG represented by fvAEPg
## encapをvlan1に変更
fvRsPathAtt2 = cobra.model.fv.RsPathAtt(fvAEPg,
tDn=u'topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2A]', instrImedcy=u'lazy',
encap=vlan1, descr=u'', mode=u'regular')

# creates a relationship between the EPG represented by fvAEPg and a Physical Domain.
# this determines what encapsulation values can be used.
fvRsDomAtt = cobra.model.fv.RsDomAtt(fvAEPg, instrImedcy=u'lazy', resImedcy=u'lazy',
encap=u'unknown', tDn=u'uni/phys-Heroes_phys')

# sets the QoS policy for the EPG represented by fvAEPg
fvRsCustQosPol = cobra.model.fv.RsCustQosPol(fvAEPg, tnQosCustomPolName=u'')
```

```
# this assigns the Bridge Domain that hosts in fvAEPg will belong to. the Bridge  
# Domain provides a flooding domain and a set of subnets the hosts can belong to.  
## Bridge Domain名を"Hero_Land"から、bridge_domainに変更  
fvRsBd = cobra.model.fv.RsBd(fvAEPg, tnFvBDName=bridge_domain)
```

```
# have the EPG represented by the fvAEPg object provide the "power_up" contract  
fvRsProv = cobra.model.fv.RsProv(fvAEPg, tnVzBrCPName=u'power_up',  
matchT=u'AtleastOne', prio=u'unspecified')
```

## 「db」 EPGを作成します

次の数行のコードは前のブロックと非常に似ていますが、オブジェクトタイプの新しいインスタンスを作成するために、各オブジェクト名がわずかに異なります。

encapとtnFvBDName引数はVLAN2とbridge\_domain変数を使用するように更新する必要があります。

更新が必要な行には、##で始まるコメントが含まれます。

```
# creates a new EPG object represented by the name fvAEPg2  
fvAEPg2 = cobra.model.fv.AEPg(fvAp, isAttrBasedEPg=u'no', matchT=u'AtleastOne',  
prio=u'unspecified', name=u'db', descr=u'')  
  
# assign a VLAN and vPC to the EPG represented by fvAEPg2  
## encapをvlan2に変更  
fvRsPathAtt3 = cobra.model.fv.RsPathAtt(fvAEPg2,  
tDn=u'topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2B]', instrImedcy=u'lazy',  
encap=vlan2, descr=u'', mode=u'regular')  
  
# assign a VLAN and a second vPC to the EPG represented by fvAEPg2  
## encapをvlan2に変更  
fvRsPathAtt4 = cobra.model.fv.RsPathAtt(fvAEPg2,  
tDn=u'topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2A]', instrImedcy=u'lazy',  
encap=vlan2, descr=u'', mode=u'regular')  
  
# creates a relationship between the EPG represented by fvAEPg2 and a Physical Domain.  
fvRsDomAtt2 = cobra.model.fv.RsDomAtt(fvAEPg2, instrImedcy=u'lazy',  
resImedcy=u'lazy', encap=u'unknown', tDn=u'uni/phys-Heroes_phys')  
  
# sets the QoS policy for the EPG represented by fvAEPg2  
fvRsCustQosPol2 = cobra.model.fv.RsCustQosPol(fvAEPg2, tnQosCustomPolName=u'')  
  
# this assigns the Bridge Domain that hosts in the fvAEPg2 will belong to  
## Bridge Domain名を"Hero_Land"から、bridge_domainに変更  
fvRsBd2 = cobra.model.fv.RsBd(fvAEPg2, tnFvBDName=bridge_domain)
```

```
# have the EPG represented by the fvAEPg2 object provide the "sql" contract  
fvRsProv2 = cobra.model.fv.RsProv(fvAEPg2, tnVzBrCPName=u'sql', matchT=u'AtleastOne',  
prio=u'unspecified')
```

## 「Web」 EPGを作成する

次の数行のコードも前のブロックと非常に似ていますが、オブジェクトタイプの新しいインスタンスを作成するために、各オブジェクト名がわずかに異なります。

encapとtnFvBDName引数はVLAN3とbridge\_domain変数を使用するように更新する必要があります。

更新が必要な行には、##で始まるコメントが含まれます。

```
# creates a new EPG object represented by the name fvAEPg3  
fvAEPg3 = cobra.model.fv.AEPg(fvAp, isAttrBasedEPg=u'no', matchT=u'AtleastOne',  
prio=u'unspecified', name=u'web', descr=u'')  
  
# have the EPG created by the previous line of code consume the "sql" contract.  
fvRsCons2 = cobra.model.fv.RsCons(fvAEPg3, tnVzBrCPName=u'sql', prio=u'unspecified')  
  
# assign a VLAN and vPC to the EPG represented by fvAEPg3  
## encapをvlan3に変更  
fvRsPathAtt5 = cobra.model.fv.RsPathAtt(fvAEPg3,  
tDn=u'topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2B]', instrImedcy=u'lazy',  
encap=vlan3, descr=u'', mode=u'regular')  
  
# assign a VLAN and a second vPC to the EPG represented by fvAEPg3
```

```

## encapsをvlan3に変更
fvRsPathAtt6 = cobra.model.fv.RsPathAtt(fvAEPg3,
tDn=u'topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2A]', instrImedcy=u'lazy',
encap=vlan3, descr=u'', mode=u'regular')

# creates a relationship between the EPG represented by fvAEPg3 and a Physical Domain.
fvRsDomAtt3 = cobra.model.fv.RsDomAtt(fvAEPg3, instrImedcy=u'lazy',
resImedcy=u'lazy', encap=u'unknown', tDn=u'uni/phys-Heroes_phys')

# sets the QoS policy for the EPG represented by fvAEPg3
fvRsCustQosPol3 = cobra.model.fv.RsCustQosPol(fvAEPg3, tnQosCustomPolName=u'')

# this assigns the Bridge Domain that hosts in the fvAEPg3 will belong to
## Bridge Domain名を"Hero_Land"から、bridge_domainに変更
fvRsBd3 = cobra.model.fv.RsBd(fvAEPg3, tnFvBDName=bridge_domain)

# have the EPG represented by the fvAEPg3 object provide the "web" contract
fvRsProv3 = cobra.model.fv.RsProv(fvAEPg3, tnVzBrCPName=u'web', matchT=u'AtleastOne',
prio=u'unspecified')

```

### 構成をコミットする

最後に、構成の変更をAPICに送信します。これは、構成要求を作成し、要求に最上位オブジェクトを追加し、変更をコミットすることによって行われます。

```

# commit the generated code to APIC
print toXMLStr(fvTenant)
c = cobra.mit.request.ConfigRequest()
c.addMo(fvTenant)
md.commit(c)

```

すべての変更が完了したら、必ずファイルを保存してください。最終的なスクリプトは、サンプルコードリポジトリにとしてadd\_application\_SOLUTION.pyに記載されています。

## 演習 5:ターミナルとコードノート

### アプリケーションを構築する

WebAryaからの出力が正しい情報で更新されると、スクリプトを実行して新しい構成をコミットする準備が整います。

### スクリプトを実行する

新しいスクリプトはPythonファイルであり、ターミナルウィンドウから実行できます。ファイルが保存されているディレクトリを参照し、プログラムを呼び出して実行するのが最も簡単です。

```

$ cd aci-learning-labs-code-samples$ sbx-intermediate-aci
$ cd sbx-intermediate-aci-00_webarya
$ ls
add_application.py
$ python add_application.py
<?xml version="1.0" encoding="UTF-8"?>
<fvTenant name='Heroes' status='created,modified'><fvAp ownerKey='' name='Save_The_Network' descr='' status='created,modified' ownerTag='' prio='unspecified'><fvAEPg isAttrBasedEPg='no' matchT='AtleastOne' name='web' descr='' status='created,modified' prio='unspecified'><fvRsPathAtt descr='' mode='regular' status='created,modified' instrImedcy='lazy' encap='vlan-210' tDn='topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2B]'></fvRsPathAtt><fvRsPathAtt descr='' mode='regular' status='created,modified' instrImedcy='lazy' encap='vlan-210' tDn='topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2A]'></fvRsPathAtt><fvRsDomAtt status='created,modified' instrImedcy='lazy' encap='unknown' tDn='uni/phys-Heroes_phys' resImedcy='lazy'></fvRsDomAtt><fvRsBd status='created,modified' tnFvBDName='Hero_Land'></fvRsBd><fvRsCustQosPol tnQosCustomPolName='' status='created,modified'></fvRsCustQosPol><fvRsProv status='created,modified' prio='unspecified' matchT='AtleastOne' tnVzBrCPName='web'></fvRsProv><fvRsCons status='created,modified' prio='unspecified' tnVzBrCPName='sql'></fvRsCons></fvAEPg><fvAEPg isAttrBasedEPg='no' matchT='AtleastOne' name='app' descr='' status='created,modified' prio='unspecified'><fvRsPathAtt descr='' mode='regular' status='created,modified' instrImedcy='lazy' encap='vlan-211' tDn='topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2B]'></fvRsPathAtt><fvRsP

```

```

athAtt descr=' ' mode='regular' status='created,modified' instrImedcy='lazy'
encap='vlan-211'
tDn='topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2A]'></fvRsPathAtt><fvRsD
omAtt status='created,modified' instrImedcy='lazy' encaps='unknown'
tDn='uni/phys-Heroes_phys' resImedcy='lazy'></fvRsDomAtt><fvRsBd
status='created,modified' tnFvBDName='Hero_Land'></fvRsBd><fvRsCustQosPol
tnQosCustomPolName=' ' status='created,modified'></fvRsCustQosPol><fvRsProv
status='created,modified' prio='unspecified' matchT='AtleastOne'
tnVzBrCPName='power_up'></fvRsProv><fvRsCons status='created,modified'
prio='unspecified' tnVzBrCPName='sql'></fvRsCons></fvAEPg><fvAEPg
isAttrBasedEPg='no' matchT='AtleastOne' name='db' descr=' ' status='created,modified'
prio='unspecified'><fvRsPathAtt descr=' ' mode='regular' status='created,modified'
instrImedcy='lazy' encaps='vlan-212'
tDn='topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2B]'></fvRsPathAtt><fvRsP
athAtt descr=' ' mode='regular' status='created,modified' instrImedcy='lazy'
encap='vlan-212'
tDn='topology/pod-1/protpaths-101-102/pathep-[Heroes_FI-2A]'></fvRsPathAtt><fvRsC
ustQosPol tnQosCustomPolName=' '
status='created,modified'></fvRsCustQosPol><fvRsDomAtt status='created,modified'
instrImedcy='lazy' encaps='unknown' tDn='uni/phys-Heroes_phys'
resImedcy='lazy'></fvRsDomAtt><fvRsProv status='created,modified'
prio='unspecified' matchT='AtleastOne' tnVzBrCPName='sql'></fvRsProv><fvRsBd
status='created,modified'
tnFvBDName='Hero_Land'></fvRsBd></fvAEPg></fvAp></fvTenant>
$
```

これで、Heroesテナントで呼び出されるアプリケーションSave\_The\_Networkができました。

## 構成を検証する

構成が機能したことを確認するには、APIC GUIにログインし、Heroes Tenantを参照して、Application Profilesディレクトリを展開します。

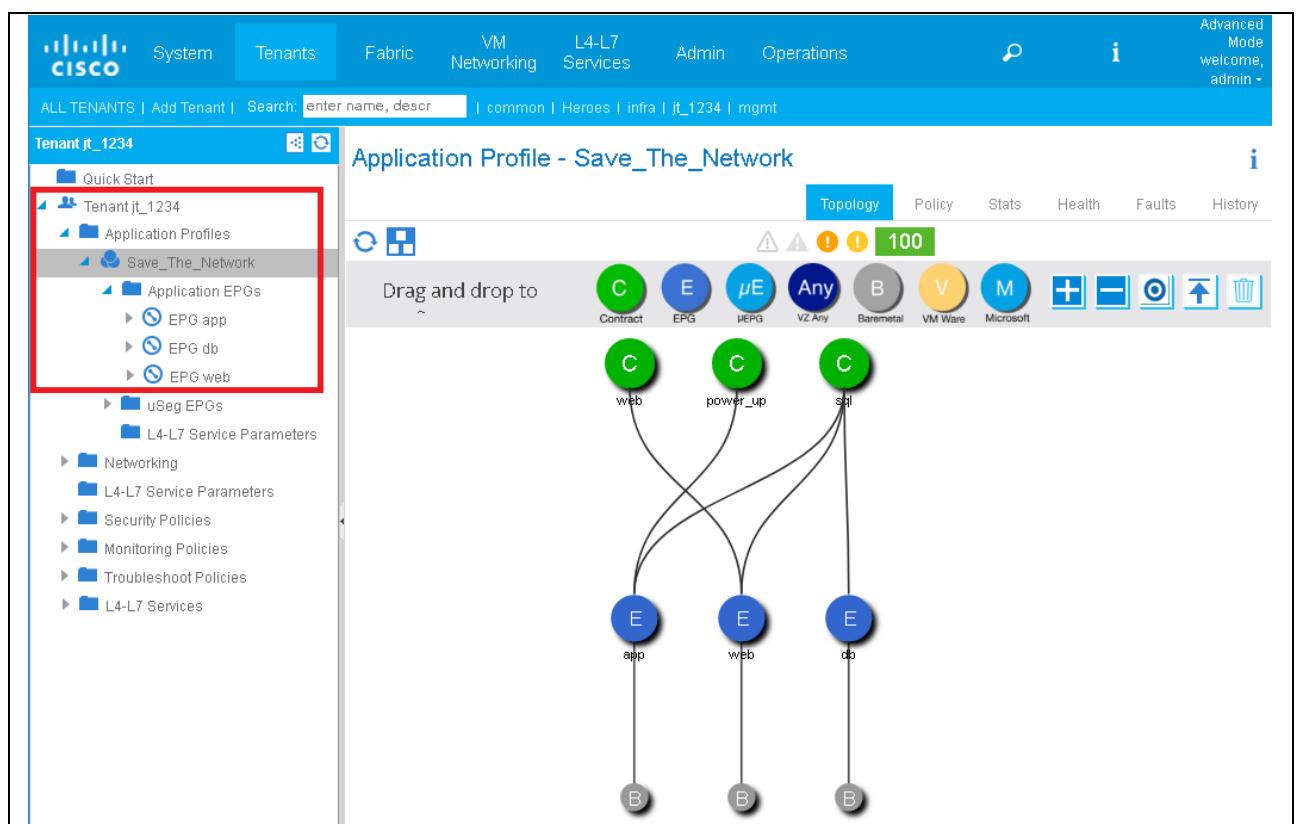


図1-46. HeroesConfigの検証

## 3.2. Cobra を使用して ACI Python スクリプトに「Bite」を追加

### 目的

完了時間：30分

- Cobra Python SDK for ACIのコンポーネントを理解する
- Cobraを使用してAPICに接続する方法を理解する
- Cobraを使用してAPICに情報を照会する方法を理解する
- Cobraを使用してPythonでACIオブジェクトを構築し、APICに送信する方法を理解する

### コードサンプル

- このラボには、演習中のコードサンプルへの参照が含まれています。  
これらはgithub.com/CiscoDevNet/aci-learning-labs-code-samplesで入手できます。
- このラボのサンプルは、sbx-intermediate-aciにあります。

## ステップ 1: Cobra 入門

「ACIプログラム応用」では、Cobra SDKが導入され、CobraがACIファブリックのプログラミングに使用される完全に機能するPythonライブラリであるという要点が紹介されました。このラボでは、「API」ラボと同様のタスクを実行して、この点を示します。

ACI Toolkit、Webaryaを使用する場合でも、POSTMANでAPI呼び出しを行う場合でも、すべての演習では、コード行を手動で編集してAPICに送信することにより、新しい構成を構築しました。将来同様の変更を加えたい場合は、更新が必要な行を覚えて、同じ手順を繰り返す必要があります。この方法は面倒で間違いがちで、変更を加える前に最小限のチェックしか提供しません。

このラボでは、PythonとCobraを使用して再利用可能なスクリプトを作成することにより、ACIのプログラマビリティ機能を拡張します。このスクリプトは、受け入れ可能な引数を一覧表示するためのヘルプメニューを提供し、構成の変更をコミットする前の基本的なチェックを提供します。まず、Cobraのmitパッケージとmodelパッケージを簡単に見てみましょう。

## ステップ 2: 「acicobra」と「acimodel」の区別

Cobraをインストールするには、2つの.whlファイルをインストールする必要があります。

- acicobra
- acimodel

### acicobra

acicobraパッケージはSDKであり、コントローラとの対話に使用されます。cobra.mit内でいくつかのモジュールとより一般的に使用されるクラスを次に示します。

- **Session**は、LoginSessionまたはCertSessionクラスのいずれかを使用してAPICとのセッションを作成するために使用されます。
- **Access**は、APICのログイン/ログアウト、およびMoDirectoryクラスを使用したクエリと構成要求の送信に使用されます。
- **Request**は、DnQueryクラスとClassQueryクラスを使用してクエリを作成するため、およびConfigRequestクラスを使用して構成リクエストを作成するために使用されます。

### acimodel

acimodelパッケージには、MITをモデル化するモジュールが含まれています。このパッケージのモジュールはcobra.modelの下にあり、リストが多すぎます。コブラは、オブジェクトモデルの1対1のマッピングです。したがって、オブジェクトモデルのすべてのクラスは、acimodelパッケージのクラスによって表されます。クラスオブジェクトを表すためにどのモジュールが必要かがすぐにわかるとは限らないため、JSON構成に基づいてAryaを使用してこれを決定する方法の例を示します。

「ACI APIのピーリングバック」では、次のような単純化されたJSON構成を確認しました。

```
{  
    "fvTenant": {  
        "attributes": {  
            "name": "Heroes",  
        },  
        "children": [  
            {  
                "fvCtx": {}  
            }  
        ]  
    }  
}
```

```

        },
        {
          "l3extOut": {}
        }
      ]
    }
}

```

`fvTenant`とその子オブジェクトは、それぞれ独自のクラスです。モジュールとクラスは、これらの名前から次のように派生しています。

- 使用されるモジュールは、最初の大文字までの文字です。
- 使用されるクラスは、名前の残りの部分です。

この例では、`cobra.model.fv`は、`Tenant`や`Ctx`クラスのためにインポートする必要、と`cobra.model.l3ext`は、`Out`クラスのために必要とされています。

より簡単なオプションは、オブジェクトの作成時に完全な名前空間を使用するため、`Arya`を使用することです。`Arya`が生成するものの例を次に示します。

```

import cobra.model.fv
import cobra.model.l3ext

fvTenant = cobra.model.fv.Tenant(...)
fvCtx = cobra.model.fv.Ctx(...)
l3extOut = cobra.model.l3ext.Out(...)

```

各オブジェクトは、インポートされたモジュールの一つに続いて`.Classname`を使用して作成されていることに注意してください。

`acimodel`パッケージはMITのみをモデル化し、そのクラスを使用して作成されたオブジェクトはAPICに存在するものを表していないことに注意することが重要です。`acicobra`パッケージのみがコントローラと対話し、APIC MITの既存のオブジェクトを表すオブジェクトを作成できます。

これは、スクリプトを使用して、またはインターパリターでの作業中に作成されたオブジェクトは、コミットが成功するまでローカルにのみ存在することを意味します。間違えて取り消す必要がある場合は、インターパリタセッションを閉じると、変更が行われないようになります。

### ステップ 3: ACI Cobra パッケージの使用

`acicobra`パッケージには、ログイン、クエリ、およびACIファブリックへの設定の送信に使用されるクラスが含まれています。ログインクラスとクエリクラスを使用して、`acicobra`パッケージの使用に慣れれます。

構成の送信は、`acimodel`パッケージでカバーされます。

### 演習 1 セッションの確立

スクリプトの最初のステップはログインです。これには、セッションモジュールとアクセスモジュールが必要です。

Pythonインターパリターを開き、次のコードブロックに従って、APICとのセッションを確立します。

資格情報ファイルを使用していない場合は、お使いの環境に適切な情報を持つ変数URL、LOGIN、PASSWORDをあなたが提供する必要があります。

```

$ python
# import classes
from credentials import *
import cobra.mit.session
import cobra.mit.access

#disable Secure Request Warning
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

URL = "https://apic1.dcloud.cisco.com"
LOGIN = "admin"
PASSWRD = "Cisco12345"

# establish a session with the APIC
auth = cobra.mit.session.LoginSession(URL, LOGIN, PASSWORD)

```

```
session = cobra.mit.access.MoDirectory(auth)
session.login()
>>>
```

このウィンドウを開いたままにして、このセッションを使用してAPICでクエリを実行します。後でセッションの有効期限が切れたというエラーを受け取った場合は、`session.login()`を再入力してください。

## 演習 2 クエリを作成する

DNレベルとクラスレベルの両方のクエリを使用してMITにクエリを実行するAPIを使用して導入された「ACI APIのピーリングバック」。Cobraは、DnQueryクラスとClassQueryクラスを使用してこれと同じ機能を提供します。クエリは、一致するすべてのオブジェクトのリストを返します。各一致には、特定のクラスがサポートする一連のプロパティがあり、これらのプロパティには`.property`を使用してアクセスできます。

### ターミナルとコードノート

この演習では、必ずPythonのコーディング端末を開き、APICとのセッションを作成するために使用したPythonインターブリターウィンドウで、次のコードスニペットに従ってください。

#### HerosテナントのDNクエリを作成する

```
import cobra.mit.request

tenant_query = cobra.mit.request.DnQuery("uni/tn-Heroes")
heroes_tenant = session.query(tenant_query)

heroes_tenant
>>>
出力は次のようにになります。
```

```
[<cobra.modelimpl.fv.tenant.Tenant object at 0x7ff60b5bb2d0>]
```

#### Heroesテナントオブジェクトで利用可能なプロパティを表示する

```
heroes = heroes_tenant[0]
dir(heroes)
```

出力は次のようにになります。

```
['__BaseMo__children', '__BaseMo__dirtyProps', '__BaseMo__dn', '__BaseMo__meta',
 '__BaseMo__modifyChild', '__BaseMo__parentDn', '__BaseMo__parentDnStr',
 '__BaseMo__parentMo', '__BaseMo__rn', '__BaseMo__setModified', '__BaseMo__setprop',
 '__BaseMo__status', '__ChildContainer', '__class__', '__delattr__', '__dict__',
 '__doc__', '__format__', '__getattribute__', '__hash__', '__init__',
 '__module__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
 '__sizeof__', '__str__', '__subclasshook__', '__weakref__', '_attachChild',
 '_children', '_delete', '_detachChild', '_dirtyProps', '_dn', '_isPropDirty',
 '_numChildren', '_parent', '_parentDn', '_resetProps', '_rn', '_setParent', '_status',
 '_childAction', '_children', '_clone', '_contextRoot', '_delete', '_descr', '_dirtyProps',
 '_dn', '_isInstance', '_isPropDirty', '_lcOwn', '_meta', '_modTs', '_monPolDn', '_name',
 '_numChildren', '_ownerKey', '_ownerTag', '_parent', '_parentDn', '_prop', '_resetProps',
 '_rn', '_status', '_uid', '_update']
```

#### オブジェクトプロパティの表示

```
heroes.name
# 'Heros'

heroes.dn
# <cobra.mit.naming.Dn object at 0x7ff60c09a910>

print(heroes.dn)
# uni/tn-Heroes

str(heroes.dn)
# 'uni/tn-Heroes'
```

## 演習 3 スコープクエリ

リクエストクラスは、結果をフィルタリングし、MITの特定の部分を含めるためのスコープクエリもサポートしています。これらのスコーププロパティは、「ACI APIのピーリングバック」で説明されているAPIメソッドの一つと関連します。

- `propFilter(query-target-filter)`は、クラス属性に基づいて結果をフィルタリングします。

- `queryTarget(query-target)`は、MITのどの部分をクエリするかを指定します。
  - `self`
  - `children`
  - `subtree`
- `classFilter(target-subtree-class)`は、`queryTarget`が「`children`」または「`subtree`」のいずれかに設定されている場合に使用して、返された子を特定の1つまたは複数のクラスのオブジェクトのみにフィルタリングできます。
  - たとえば、アプリケーションプロファイルをフィルタリングするには、`fvAp`を使用します。
- サブツリー(`rsp-subtree`)は、取得するサブツリーの量を指定します。
  - `no`
  - `children`
  - `full`
- `subtreeClassFilter(rsp-subtree-class)`は、応答に含めるサブツリークラスをフィルタリングします。
- `subtreeInclude(rsp-subtree-include)`は、サブツリーに含める情報のタイプを指定します
  - `audit-logs`
  - `event-logs`
  - `faults`
    - ✧ `faults,no-scoped`
  - `health`

次のいくつかのコードブロックは、クラスクエリの使用方法を示し、スコープが返される結果にどのように影響するかを示しています。まず、APICで構成されているすべてのアプリケーションを確認します。独自のPythonインターフリターに従ってください。

#### すべてのアリプロファイルのクラスクエリを作成する

```
app_query = cobra.mit.request.ClassQuery('fvAp')

apps = session.query(app_query)
apps
# [<cobra.modelimpl.fv.ap.Ap object at 0x7f3c3c87b6d0>, <cobra.modelimpl.fv.ap.Ap object at 0x7f3c3c872a50>,
# <cobra.modelimpl.fv.ap.Ap object at 0x7f3c3c872b90>]
for app in apps:
    print(app.name)

# Example_App
# access
# Save_The_Planet
# default
# Rescue
# Chaos
```

#### 「Save\_The\_Planet」という名前のアプリケーションにクエリのスコープを設定します

```
# set the property filter to only return the app named "Save_The_Planet"
app_query.propFilter = 'eq(fvAp.name, "Save_The_Planet")'
save_the_planet_app = session.query(app_query)
save_the_planet_app
# [<cobra.modelimpl.fv.ap.Ap object at 0x7f3c3cd310>]

save_the_planet_app[0].name
# Save_The_Planet

# save_the_planet_app did not return any child objects for the application
save_the_planet_app[0].numChildren
# 0
```

### 3.3. 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します。

課題：アプリケーションヘルスダッシュボードを作成する  
目的

完了時間：45分

- 実際の運用ユースケースでCobraSDKを使用する
- テナントの状態と障害情報を表示するWebダッシュボードを作成します

#### コードサンプル

- このラボには、演習中のコードサンプルへの参照が含まれています。  
これらはgithub.com/CiscoDevNet/aci-learning-labs-code-samplesで入手できます。
- このラボのサンプルは、sbx-intermediate-aciにあります。

#### ステップ 1: ミッションの概要

スーパーヒーローと悪役(SnV)が成長し続けるにつれて、インシデントを迅速に切り分けて解決する必要性が高まっています。新しいデータセンターにACIを導入したため、上司はプログラミング機能を活用してこれらの要件を満たすことにしました。

SnVのアプリケーションとそのヘルススコアを表示するアプリケーションヘルスダッシュボードを作成するように求められました。ACIのCobraパッケージを使用して、ファブリックにクエリを実行し、SnVのすべてのアプリケーションの状態を報告し、アプリケーションがクリックされたときに障害を報告することにしました。

#### 資格情報ファイル

1. credentials.pyファイルを編集します。

ファイルの場所:

c:\Users\demouser\aci-learning-labs-code-samples\sbx-intermediate-aci\sbx-intermediate-aci-mission1\_health-dashboard\dashboard\credentials.py

```
URL = 'https://apic1.dcloud.cisco.com'  
LOGIN = 'admin'  
PASSWORD = 'Cisco12345'
```

#### ステップ 2: ダッシュボードアプリケーション

##### ターミナルとコードノート

この演習では、必ずPythonのコーディング端末を開き、このラボのコードサンプルのディレクトリを入力してください。

```
> aci-learning-labs-code-samples\sbx-intermediate-aci\sbx-intermediate-aci-mission1_health-dashboard
```

サンプルコードリポジトリには、dashboardという名前のディレクトリがあります。そのディレクトリ内には、SnVテナントの下にあるすべてのアプリケーションの状態を監視するためのFlask(Python Web Framework)アプリケーションがあります。

##### views.pyを修正する

dashboardディレクトリから、appサブディレクトリを開きます。このディレクトリ内には、ファイルviews.pyがあります。このファイルには、アプリケーションのリスト、それらの現在のヘルススコア、およびアプリケーションに関連する障害を収集するために使用される関数が含まれています。いくつかの行には、正しいコードに置き換える必要がある**SET ME**と書かれた部分があります。また、コードブロックの下にいくつかのヒントがあります。

「Cobra」ラボには、Cobraでのクエリの作成とスコープに関するディスカッションが含まれています。「ACI API」ラボでは、スコープフィルターオプションの詳細について説明しています。

```
from flask import render_template, redirect, abort, request, url_for, jsonify  
from app import app  
import cobra.mit.access  
import cobra.mit.request  
import cobra.mit.session  
from credentials import *  
import requests, json, sys
```

```

def get_healthscore():
    session = aci_login()

    app_query = cobra.mit.request.DnQuery('uni/tn-SnV')
    app_query.queryTarget = SET ME
    app_query.classFilter = SET ME
    app_query.subtreeInclude = SET ME

    apps = SET ME
    health_dict = {}

    for app in apps:
        for health in app.children:
            health_dict[app.name] = int(health.cur)

    return health_dict

def get_faults(app_name):
    session = aci_login()

    fault_query = cobra.mit.request.DnQuery('uni/tn-SnV/ap-{}'.format(app_name))
    fault_query.queryTarget = 'subtree'
    fault_query.subtreeInclude = 'faults,no-scoped'
    fault_query.orderBy = 'faultInfo.severity|desc'
    fault_query.page = 0
    fault_query.pageSize = 15

    faults = session.query(fault_query)
    faults_dict = {'faults': []}

    for fault in faults:
        if fault.lc == 'retaining':
            fault_dict = {
                'Acknowledged': fault.ack,
                'Affected': 'Issue No Longer Exists',
                'Description': fault.descr,
                'Time': fault.created,
                'Life Cycle': fault.lc
            }
        else:
            fault_dict = {
                'Acknowledged': fault.ack,
                'Affected': fault.affected,
                'Description': fault.descr,
                'Time': fault.created,
                'Life Cycle': fault.lc
            }

        faults_dict['faults'].append(fault_dict)

    return faults_dict

def aci_login():
    auth = cobra.mit.session.LoginSession(URL, LOGIN, PASSWORD)
    session = cobra.mit.access.MoDirectory(auth)
    session.login()

    return session

@app.route('/', methods=['GET', 'POST'])
def index():
    health_dict = get_healthscore()
    return render_template("index.html", health_dict=health_dict)

```

```

@app.route('/healthscore_update', methods=['POST'])
def healthscore_update():
    if request.method == 'POST':
        return jsonify(get_healthscore())

@app.route('/fault_update', methods=['POST'])
def fault_update():
    if request.method == 'POST':
        return jsonify(get_faults(request.form['app']))

```

### get\_healthscore()関数

更新が必要な行とヒントを次に示します。

1. app\_query.queryTarget = **SET ME**
- 子オブジェクトのみを返すように設定する必要があります。
2. app\_query.classFilter = **SET ME**
- APIがアプリケーションプロファイルを識別するために使用するクラス名である必要があります。
3. app\_query.subtreeInclude = **SET ME**
- ヘルススコアを含めるために使用される文字列である必要があります。
4. apps = **SET ME**
- app\_queryオブジェクトのAPICをクエリするために使用される関数である必要があります。

解決策は次のステップにあります。

## ステップ 3 解決

### ターミナルとコードノート

次のコードは完成したviews.pyファイルです。また、コードディレクトリ内にviews\_SOLUTION.pyとして保存されます

```

from flask import render_template, redirect, abort, request, url_for, jsonify
from app import app
import cobra.mit.access
import cobra.mit.request
import cobra.mit.session
from credentials import *
import requests, json, sys

def get_healthscore():
    session = aci_login()

    app_query = cobra.mit.request.DnQuery('uni/tn-SnV')
    app_query.queryTarget = 'children'
    app_query.classFilter = 'fvAp'
    app_query.subtreeInclude = 'health'

    apps = session.query(app_query)
    health_dict = {}

    for app in apps:
        for health in app.children:
            health_dict[app.name] = int(health.cur)

    return health_dict

def get_faults(app_name):
    session = aci_login()

    fault_query = cobra.mit.request.DnQuery('uni/tn-SnV/ap-{}'.format(app_name))
    fault_query.queryTarget = 'subtree'
    fault_query.subtreeInclude = 'faults,no-scoped'
    fault_query.orderBy = 'faultInfo.severity|desc'
    fault_query.page = 0
    fault_query.pageSize = 15

```

```

faults = session.query(fault_query)
faults_dict = {'faults': []}

for fault in faults:
    if fault.lc == 'retaining':
        fault_dict = {
            'Acknowledged': fault.ack,
            'Affected': 'Issue No Longer Exists',
            'Description': fault.descr,
            'Time': fault.created,
            'Life Cycle': fault.lc
        }
    else:
        fault_dict = {
            'Acknowledged': fault.ack,
            'Affected': fault.affected,
            'Description': fault.descr,
            'Time': fault.created,
            'Life Cycle': fault.lc
        }

    faults_dict['faults'].append(fault_dict)

return faults_dict

def aci_login():
    auth = cobra.mit.session.LoginSession(URL, LOGIN, PASSWORD)
    session = cobra.mit.access.MoDirectory(auth)
    session.login()

    return session

@app.route('/', methods=['GET', 'POST'])
def index():
    health_dict = get_healthscore()
    return render_template("index.html", health_dict=health_dict)

@app.route('/healthscore_update', methods=['POST'])
def healthscore_update():
    if request.method == 'POST':
        return jsonify(get_healthscore())

@app.route('/fault_update', methods=['POST'])
def fault_update():
    if request.method == 'POST':
        return jsonify(get_faults(request.form['app']))

```

## ダッシュボードを表示する

### ターミナルとコードノート

作業views.pyファイルができたので、先に進んでdashboardを見てください。

まず、run.pyダッシュボードディレクトリでファイルを実行してアプリケーションを起動します。

```

$ pwd
your_working_dir/aci-learning-labs-code-samples/sbx-intermediate-aci/sbx-intermediate-aci-mission1_health-dashboard/dashboard
$ ls
app credentials.py requirements.txt run.py
$ python run.py
* Sering Flask app "app" (lazy loading)
* Environment: production
WARNING: Th is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.

```

```
* Debug mode: on
* Restarting with stat
* Debugger is active!
* Debugger pin code: 102-464-421
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

ダッシュボードはポート5000を使用するため、ブラウザをhttp://localhost:5000にアクセスして、アプリケーションヘルスダッシュボードを表示します。

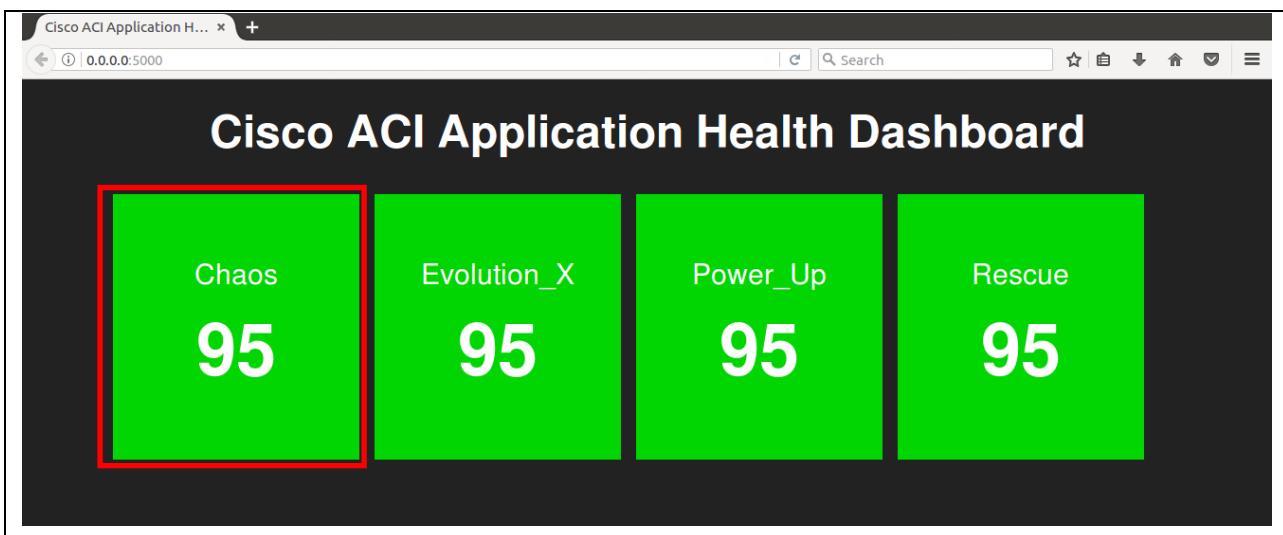


図1-47. アプリケーションヘルスダッシュボード

アプリケーションの1つをクリックして、それに関連する障害を表示します。詳細については、ボックスをクリックしてください。

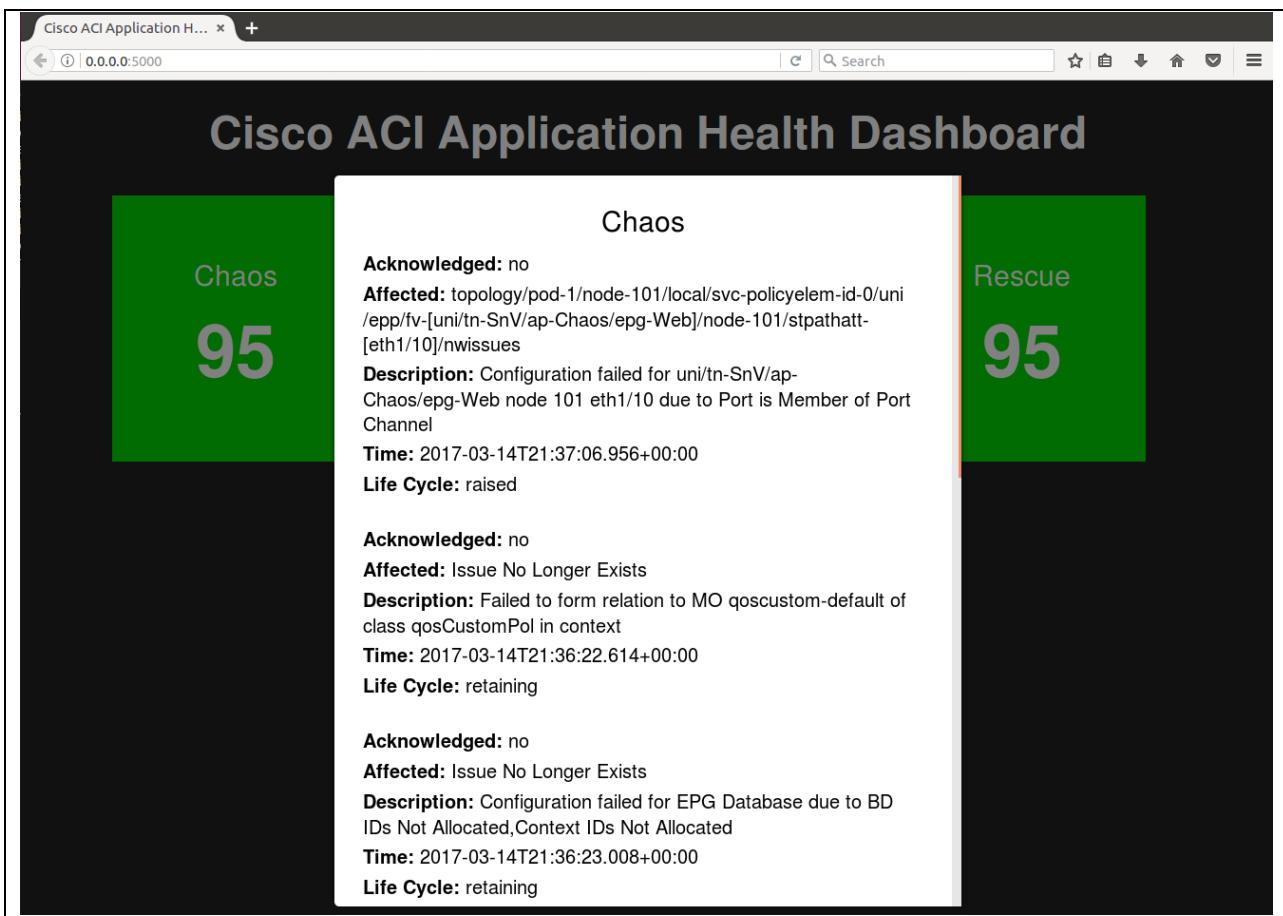


図1-48. アプリケーションの障害

これで、上司に引き渡す準備ができているアプリケーションヘルスダッシュボードが正常に作成されました。  
お疲れ様でした！

## 4. ACI と Ansible の紹介

「AnsibleCompletedでCiscoACIの自動化を開始」

ACIでAnsibleを使用し、コードとしてテナントを作成する方法について説明

「Ansibleが完成したアプリケーションとネットワークポリシー」

Ansibleモジュールを使用してACIアプリケーションとネットワークポリシーを制御

「Ansibleが完成したコードとしてACI用にまとめる」

Playbookを作成して、AnsibleでACI構成を完全に管理

## 4.1. Ansible で Cisco ACI の自動化を実行

### 概要

Cisco Application Centric Infrastructure(ACI)には、APIを使用してプログラムでACIファブリックを構築および運用するのに役立つツールがいくつかあります。これらのツールには、APIインスペクター、ACI Cobra SDK、ACIツールキットなどがあります。これらのツールはすべて、他のDevNet Learning Labsでカバーされています。

この学習モジュールのラボでは、Ansibleを使用してACIを自動化する最新の方法を使用して説明します。Ansible、特にAnsibleのACIモジュールを使用する利点は、カスタムスクリプトを作成および開発する必要がなくなることです。Ansibleモジュールは基本的に、特定のタスクを実行するために作成されたスクリプトです。これにより、APIリクエストの調査、適切なAPIリクエストを行うためのコードの開発、コードのテストとデバッグにかかる時間を節約できます。

### 目的

- セットアップとAnsibleおよびACI環境
- ACI Ansibleモジュールを理解する
- Ansibleを使用してACIテナントを作成する

## ステップ 1 : Ansible および ACI 環境をセットアップする

### Ansible ACIモジュールの概要

このラボは、Linuxマシンで実行します。

ACIモジュールがインストールされたAnsibleバージョンができたので、モジュール自体についてもう少し理解する必要があります。

ACI Ansibleモジュールは、Ansibleモジュールの動作と同じように機能します。リモートデバイスへの接続と構成に使用される固定パラメータがあります。この場合、APICのアプリケーションポリシーインフラストラクチャコントローラです。ACIモジュールはべき等であり、モジュールのパラメータが現在存在するものと異なる場合にのみ構成をプッシュし、モジュールの実行のステータスと行われた変更がある場合はそれを返します。

エンドデバイスが目的の状態にあることを確認するために、一般的なAnsible言語を使用します。Ansibleモジュールの目的は、エンドデバイスに変更を適用することではなく、モジュールに渡されるパラメータに従ってエンドデバイスが目的の状態にあることを確認することです。

## ステップ 2: Ansible ACI Playbook を確認する

ACIテナントモジュールは幅広いACIテナント構成をサポートしますが、テナントネットワーク、ポリシー、およびEPGの管理に必要なコアモジュールを紹介します。

このラボでは、各モジュールの目的、固有のパラメータ、および期待される出力について説明します。このモジュールの最後に、テナント構成が存在することを確認するPlaybookを作成して使用します。以下のACI Toolkitの図は、このAnsible Playbookが構成して存在することを確認する構成です。

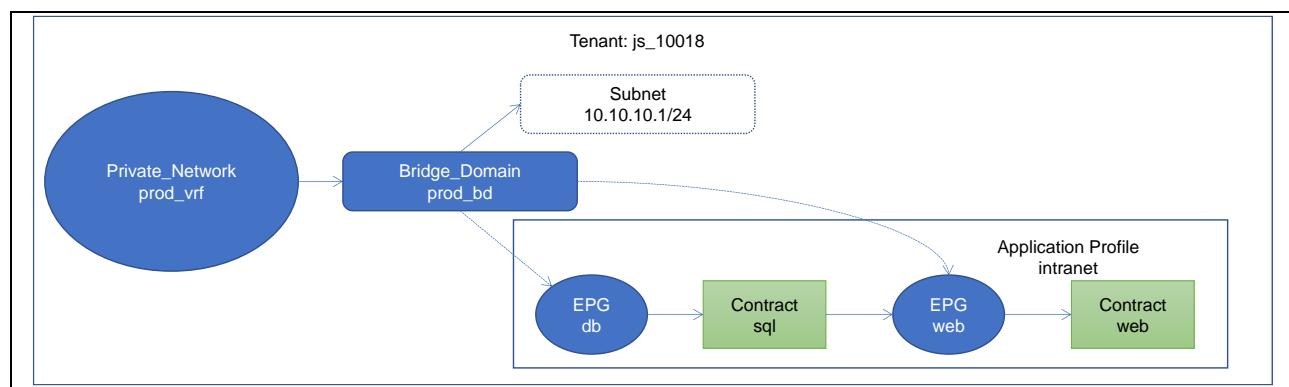


図1-49. Ansibleの例

## 演習 1: Ansible ACI Playbook を表示する

この演習では、この演習の最初に複製されたリポジトリに付属のinventoryファイルを使用しています。inventoryファイルは、aci\_ansible\_learning\_labs\_code\_samples/intro\_moduleディレクトリにあります。

```
[apic:vars]
username=admin
password=ciscopsdt
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。

今回の環境用に、以下のように変更を行います。

```
[apic:vars]
username=admin
password=Cisco12345
ansible_python_interpreter="/usr/bin/env python"
```

```
[apic]
apic1.dcloud.cisco.com
```

### pythonの保存場所を確認

inventoryファイルの変数ansible\_python\_interpreterを、仮想環境のpythonインターペリターに更新する必要があります。Pythonインターペリターへのパスを確認するには、プロンプトで次のように入力します。

### which python

アクティビ化されたPython仮想環境では、Pythonパスはこれと非常によく似ています。

```
/root/code/aci_ansible_learning_labs/intro_module/venv/bin/python
```

### テナントモジュールの使用

このaci\_tenantモジュールは、APICでテナントを管理するために使用されます。このモジュールには、注意する必要のある固有のパラメータが1つだけあります。

- tenant : これは、APICから構成を管理または取得するテナントの名前です。

注 : テナントはすべてのテナント関連オブジェクトのルート親であるため、tenantパラメータは他のすべてのテナントモジュールで使用されます。

このラボの最初に複製したディレクトリaci\_ansible\_learning\_labs\_code\_samples/intro\_module内のファイル01\_aci\_tenant\_pb.ymlを開きます。このPlaybookの最初で唯一のプレイは、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されます。

このプレイのタスクは、apic1.dcloud.cisco.comというapicグループ内のすべてのホストに対して実行されます。

```
---
- 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"
```

最初のタスク「ENSURE APPLICATION CONFIGURATION EXISTS[アプリケーションのテナントが存在することを確認する]」では、aci\_tenantモジュールを使用します。パラメータのほとんどは、APICに接続す

そのためのものではなく、tenantそしてdescriptionは、実際のコンフィギュレーションを実行するために使用されます。モジュールはべき等であるため、テナントが存在しない場合、またはテナントが存在する場合のみ変更が加えられますが、異なる説明が構成されていることに注意してください。

説明は「Tenant Created Using Ansible [Ansibleを使用して作成されたテナント]」として静的に定義されていますが、テナントの名前には変数が使用され、Playbookを実行するときにインタラクティブに入力します。これは、vars\_promptと呼ばれるAnsibleの再生属性を利用して行われます。

## 演習 2: Ansible ACI Playbook を実行する

- 01\_aci\_tenant\_pb.ymlを含むディレクトリにいることを確認してください。

Playbookでテナントの名前の入力を求められたら、イニシャルと電話番号(下4桁)を使用して名前を作成します。たとえば、電話番号1234のトップアウト次郎さんは、以下に示すように、テナント名jt\_1234を作成します。

正しいディレクトリにいるときに、Playbookを実行します。

```
$ 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
$
```

これは一意のテナントである必要があります。したがって、ok = 1およびchanged = 1の結果が得られるはずです。

APICをチェックして、新しいテナントが存在することを検証することもできます。

- Tenantsをクリックします
- 作成したTenantをダブルクリックして開きます

Name	Description	Bridge Domains	VRFs	EPGs	Health Score
common		1	2	0	100
Heroes		1	1	3	95
infra		1	1	1	100
jt_1234	Tenant Created Using Ans...	0	0	0	100
mgmt		1	2	0	100
SnV		1	1	8	95

図1-50. オープンテナント

テナントを開くと、左側のナビゲーションにその子が表示されます。これらを使用して、他のAnsible ACI ラボ全体で期待どおりにPlaybookが機能することを検証できます。

The screenshot shows the Cisco Application Centric Infrastructure (ACI) tenant management interface. The top navigation bar includes tabs for System, Tenants, Fabric, VM Networking, L4-L7 Services, Admin, Operations, and a user dropdown for 'Advanced Mode' and 'admin'. Below the navigation is a search bar with placeholder text 'Search: enter name, descr'.

The main content area is titled 'Tenant - jt\_1234'. On the left, a sidebar menu under 'Tenant jt\_1234' lists options like Quick Start, Application Profiles, Networking, L4-L7 Service Parameters, Security Policies, Monitoring Policies, Troubleshoot Policies, and L4-L7 Services. The 'Application Profiles' option is currently selected.

The central dashboard is titled 'Health' with a score of 100. It includes a 'Zoom' button with options for '1H', '1D', and 'All', and a 'Time' slider. A message 'No stats data to display...' is shown below the graph area.

To the right of the dashboard, there are two tables:

- Fault Counts By Domain**: A table showing fault counts across various domains. All counts are 0.

Fault Level	!	!	!	!
SYSTEM WIDE	0	0	0	0
Access	0	0	0	0
External	0	0	0	0
Framework	0	0	0	0
Infra	0	0	0	0
Management	0	0	0	0
Security	0	0	0	0
Tenant	0	0	0	0

- Fault Counts By Type**: A table showing fault counts by type. All counts are 0.

Fault Type	!	!	!	!
System	0	0	0	0
Access	0	0	0	0
External	0	0	0	0
Framework	0	0	0	0
Infra	0	0	0	0
Management	0	0	0	0
Security	0	0	0	0
Tenant	0	0	0	0

図1-51. テナントの子供

### お疲れ様です！

すばらしい仕事です。次のラボでは、Ansibleを使用してアプリケーションとネットワークポリシーを管理する方法を見ていきます。

## 4.2. Ansible で Application と Network Policy

### 概要

Cisco Application Centric Infrastructure(ACI)には、APIを使用してプログラムでACIファブリックを構築および運用するのに役立つツールがいくつかあります。これらのツールには、APIインスペクター、ACI Cobra SDK、ACIツールキットなどがあります。これらのツールはすべて、他のDevNet LearningLabsでカバーされています。

この学習モジュールのラボでは、Ansibleを使用してACIを自動化する最新の方法を使用して説明します。Ansible、特にAnsibleのACIモジュールを使用する利点は、カスタムスクリプトを作成および開発する必要がなくなることです。Ansibleモジュールは基本的に、特定のタスクを実行するために作成されたスクリプトです。これにより、APIリクエストの調査、適切なAPIリクエストを行うためのコードの開発、コードのテストとデバッグにかかる時間を節約できます。

### 目的

- ACIテナントVRFを作成する
- ACIテナントブリッジドメインを作成する
- ACIテナントブリッジドメインサブネットを作成する
- ACIテナントフィルターとコントラクトを作成する
- ACIテナントアプリケーションプロファイルを作成する
- ACIテナントエンドポイントグループを作成する

### ステップ 1 : ACI テナントネットワークモジュールを確認する

テナントネットワーク構造に関連するいくつかのモジュールがありますが、アプリケーションにネットワーク接続を提供するために必要な三つに焦点を当てます。

- aci\_vrf
- aci\_bd
- aci\_bd\_subnet

これらの各モジュールについて説明し、Ansibleを使用して三つのタスクすべてを実行し、適切なネットワークオブジェクトが存在することを確認します。

注：続行する前に、コードサンプルディレクトリ

aci\_ansible\_learning\_labs\_code\_samples/intro\_moduleにいることを確認し、インストールしたPython仮想環境でAnsibleがアクティビ化されていることを確認してください。

### ステップ 2 : ACI Ansible ネットワークモジュールを確認する

#### VRFモジュール

このaci\_vrfモジュールは、APICでテナントVRFを管理するために使用されます。このモジュールには、注意すべき3つの重要なパラメータがあります。

- **vrf** : 新しいVRFまたはコンテキストに付ける名前。
- **policy\_control\_preference** : VRFがセキュリティポリシーを実施するかどうかを決定します
- **policy\_control\_direction** : ポリシーがスイッチからの入力または出力で適用されるかどうかを決定します。

1. 02\_aci\_tenant\_network\_pb.ymlファイルを開き、最初のタスク(12行目から21行目)を確認します。

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"
```

APICによって提供されるデフォルト値を使用するだけなので、このタスクにはpolicy\_controlパラメータは含まれません。

このタスクでは、変数vrfを使用します。これは(Playbookの実行時に)、Ansibleのextra-vars機能を使用して変数をPlaybookに提供する方法を示します。このアプローチの利点は、別のVRFを使用するたびにファイルを手動で編集しなくても、Playbookを再利用できることです。

次のタスクでは、関連するブリッジドメインを追加する方法を見ていきます。

## ブリッジドメインモジュール

このaci\_bdモジュールは、ブリッジドメインを管理するために使用されます。このモジュールにはいくつかのパラメータがあります。ここにもっと重要なことがあります：

- **bd**: ブリッジドメインの名前。
- **vrf**: ブリッジドメインに関連付けられているサブネットが属するVRF。
- **scope**: サブネットを**public**(外部にアドバタイズ)、**private**(VRFでのみ使用可能)、または共有(複数のVRF間で使用)と見なすかどうかを決定します。

デフォルト値scopeが**private**であることを指摘するのは重要です。VRFまたはACIファブリックの外部でネットワークに到達できるようにする場合は、スコープを**public**に明示的に設定するPlaybookを作成する必要があります。

2. 02\_aci\_tenant\_network\_pb.ymlで、ファイル2番目のタスク(23行目から33行目)を調べます。

```
- 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"
```

このタスクは前のタスクと同じ変数vrfを使用しています。これは、両方のタスクで同じVRFを使用していることを確認するために重要です。

bdパラメータに導入された新しい変数もあります。これは、bd変数が存在しない場合のデフォルトも提供します。デフォルトの変数値を指定すると、Playbookの実行の大部分で同じ値を使用したいが、場合によってはそのデフォルト値をオーバーライドしたい場合に便利です。

## サブネットモジュール

このaci\_bd\_subnetモジュールは、ブリッジドメインに属するサブネットを管理するために使用されます。サブネットはブリッジドメインの子オブジェクトであるbdため、このモジュールでもパラメータが使用されます。より一般的なパラメータは次のとおりです。

- **gateway**: サブネット上のホストのゲートウェイアドレス。
- **subnet\_mask**: サブネットに関連付けられたサブネットマスク。
- **scope**: サブネットを外部にアドバタイズする(パブリック)か、VRFにプライベートに保つ(プライベート)か、他のテナントと共有する(共有)かを決定します。

このモジュールはもサポートしますsubnet\_nameが、これはAPICがサブネットオブジェクトを識別するために使用するルートキーではありません。

3. 02\_aci\_tenant\_network\_pb.ymlで、ファイル3番目のタスク(35行目から46行目)を調べます。

```
- 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"
```

## ステップ3：ACI Ansible テナントネットワークモジュールを実行する

Ansible Playbookのテナントネットワークモジュールに慣れてきたので、次はPlaybookを実行します。

このモジュールでは最初のラボでは、テナントを作成しました。このPlaybookを実行すると、テナント用の新しいVRF、ブリッジドメイン、およびサブネットが作成されます。

Ansibleでは、`--extra-vars`または`-e`フラグを使用して、コマンドラインで変数をPlaybookに渡すことができます。形式は、`--extra-vars "var1=value1 var2=value2"`です。ここで、`var1`と`var2`はプレイブックで使用される変数であり、`value1`と`value2`はこの実行に必要な値です。このPlaybookでは、「`vrf = prod_vrf`」を使用し、`bd`変数にデフォルトの`prod_db`を使用させます。

コマンドラインで次のコマンドを入力します：

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

前のPlaybookで使用したのと同じテナント名を使用してください!!

```
$ 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  
  
$
```

APIC GUIでテナントに戻り、次の手順に従ってネットワークが存在することを確認します。

- 「Networking」 フォルダを展開します
- 「Bridge Domains」 フォルダを展開して、作成した`prod_db`ブリッジドメインを表示します。
- 「`prod_db`」 フォルダを展開します
- 「Subnets」 フォルダを展開して、作成したサブネットを表示します
- 「VRFs」 フォルダを展開して、作成したVRFを表示します

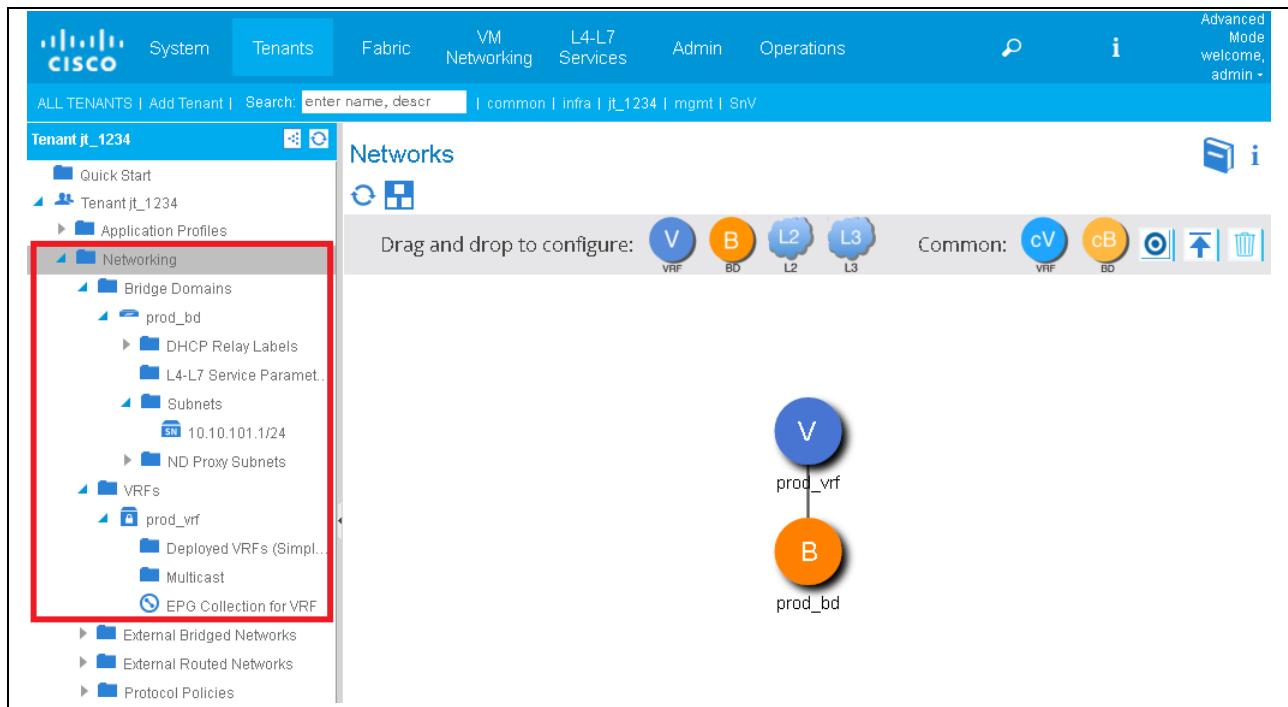


図1-52. ネットワーク構成を検証する

## ステップ 4 : ACI Ansible テナントポリシー/Contract モジュールを確認する

アプリケーションのエンドポイントグループのポリシーを構築するために必要な五つのモジュールは次のとおりです。

- aci\_filter
- aci\_filter\_entry
- aci\_contract
- aci\_contract\_subject
- aci\_contract\_subject\_to\_filter

1. **03\_aci\_tenant\_policies\_pb.yml**ファイルを開き、各モジュールについて説明し、Playbookで提供されている例を確認します。

### フィルタおよびフィルタエントリモジュール

このaci\_filterモジュールは、APICでフィルタオブジェクトaci\_filter\_entryを管理するために使用され、フィルタに関連付けられているネットワークプロトコルエントリを管理するために使用されます。

これらのモジュールには両方ともfilter、特定のフィルタを識別するために使用されるパラメータがあります。さらに、aci\_filter\_entry次のオプションとしてのモジュール：

- **entry** : フィルターエントリの名前
- **ether\_type** : オブジェクトのイーサネットタイプを設定します。「ip」が最も一般的なオプションです。
- **ip\_protocol** : ether\_typeが「ip」の場合に使用可能で、オブジェクトのIPプロトコルを決定するために使用されます。最も一般的なオプションは、「tcp」、「udp」、および「icmp」です。
- **dst\_port\_start** : TCPおよびUDPフィルターエントリで使用可能で、宛先ポート範囲の開始ポートを設定するために使用されます
- **dst\_port\_end** : TCPおよびUDPフィルターエントリで使用可能で、宛先ポート範囲の終了ポートを設定するために使用されます

2. Playbookの最初のタスクaci\_filterとaci\_filter\_entryタスク(12行目から46行目)を表示します。

```
- 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

```

これらのタスクは、`with_items`と呼ばれるタスク属性を使用します。これにより、タスクはリストをループして、複数の構成アイテムをチェックできます。アプリケーションに存在する必要のある二つのフィルタがあるため、最初のタスクには、`filter`パラメータに入力される二つの「項目」があります。

2番目のタスクには、変数値を持つ複数のパラメータがあるため、各フィールドにマッピングを提供して、どのパラメータが特定の値を取得するかを識別します。ACIは各TCP/UDPフィルターエントリをポートの範囲として扱いますが、エントリの範囲は1であるため、`dest_port_start`との両方に同じ値を使用できます `dest_port_end`。フィルタは最初のタスクのフィルタにマップされていることに注意してください。このタスクでは、これらの各フィルタに正しいポートフィルタエントリが割り当てられていることを確認します。

## Contract[コントラクト]およびContract Sub Module[コントラクトサブジェクトモジュール]

3. Playbookのタスク`aci_contract`と`aci_contract_subject`タスク(48行目から77行目)を表示します。

`aci_contract`モジュールは、コントラクトを管理するために使用され、`aci_contract_subject`モジュールは、主コントラクトに関連付けられているかのフィルタを管理するために使用されます。

`aci_contract`モジュールで利用可能な二つのより重要なオプションは次のとおりです。

- **contract** : Contractオブジェクトの名前
- **scope** : コントラクトの範囲を決定します。オプションは、「context」、「application-profile」、「tenant」、および「global」です。コントラクトのスコープは、関連するEPGによってコントラクトがどのように提供および消費されるかを決定し、コントラクトを効率的に使用する方法です。

「コンテキスト」は、コントラクト範囲のVRFと同じです

このモジュールには、`contract`パラメータに加えて、次の`aci_contract_subject`2つのパラメータがあります。

- **subject** : コントラクト主体の名前
- **reverse\_filter** : コントラクトはステートレスであるため、ACIファブリックがポートを反転して元のホストに戻るトラフィックを許可するかどうかを決定します。このパラメータは、Playbookには表示されていません。

```

- name: ENSURE TENANT CONTRACTS EXIST
  aci_contract:
    host: "{{ inventory_hostname }}"
    username: "{{ username }}"
    password: "{{ password }}"
    state: "present"
    validate_certs: False
    tenant: "{{ tenant }}"
    contract: "{{ item }}"
    scope: "context"
    description: "Contract Created Using Ansible"
  with_items:

```

```

- "web"
- "sql"

- name: ENSURE CONTRACTS HAVE CONTRACT SUBJECTS
  aci_contract_subject:
    host: "{{ inventory_hostname }}"
    username: "{{ username }}"
    password: "{{ password }}"
    state: "present"
    validate_certs: False
    tenant: "{{ tenant }}"
    contract: "{{ item.contract }}"
    subject: "{{ item.subject }}"
  with_items:
    - contract: "web"
      subject: "https"
    - contract: "sql"
      subject: "sql"

```

これらの最後の二つのタスクは、前の二つのタスクと非常によく似ています。aci\_contract\_subjectタスクのsubject\_namesは、最初のタスクのフィルタにマップされることに注意することが重要です。これにより、コントラクトが適切なフィルタに関連付けられます。

## ステップ 5 : ACI Ansible テナントポリシー/コントラクトモジュールを実行する

Ansibleには、コマンドシェルから呼び出すときに設定できるさまざまなレベルの冗長性があります。通常、Playbookの実行からの完全な戻り値を確認する必要があります。-vvv Playbook呼び出しに追加すると、モジュールの戻りデータがコマンドシェル(stdout)に出力されます。03\_aci\_tenant\_policies\_pb.yml以前のPlaybookと同じようにを呼び出しますが、今回は結果を確認するために冗長性を高めます。

コマンドラインで次のコマンドを入力します。

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

以前のPlaybookで使用したのと同じテナント名を使用してください

-vvvオプションを使用すると、かなりの量の出力があり、そのほとんどが以下の出力から省略されています。ただし、出力を確認してください。問題のトラブルシューティングを行うときに非常に役立ちます。

```
$ 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,
  "config": {
    "vzFilter": {
      "attributes": {
        "descr": "Filter Created Using Ansible",
        "name": "https"
      }
    },
    "error_code": 0,
    "error_text": "Success",
    "existing": [],
    "failed": false,
    "filter_string": "?rsp-prop-include=config-only",
    "imdata": [],
    "invocation": {
      "module_args": {
        "descr": "Filter Created Using Ansible",
        "description": "Filter Created Using Ansible",
        "name": "https"
      }
    }
  }
}
```

```

        "filter": "https",
        "host": "apic1.dcloud.cisco.com",
        "hostname": "apic1.dcloud.cisco.com",
        "method": null,
        "password": "VALUE_SPECIFIED_IN_NO_LOG_PARAMETER",
        "protocol": "https",
        "state": "present",
        "tenant": "jt_1234",
        "timeout": 30,
        "use_proxy": true,
        "use_ssl": true,
        "username": "admin",
        "validate_certs": false
    }
},
"item": "https",
"method": "POST",
"proposed": {
    "vzFilter": {
        "attributes": {
            "descr": "Filter Created Using Ansible",
            "name": "https"
        }
    }
},
"response": "OK (30 bytes)",
"status": 200,
"totalCount": "0",
"url": "https://apic1.dcloud.cisco.com/api/mo/uni/tn-[jt_1234]/flt-[https].json"
}

```

..  
(出力省略)

```

PLAY RECAP ****
apic1.dcloud.cisco.com : ok=5    changed=5    unreachable=0    failed=0
$
```

最初のプレイから最初のタスクだけを示しましたが、モジュールの実行では、実行されるタスクごとに同じタイプのデータが表示されます。返された結果を見ると、返されたデータは、このラボで以前に説明したものと一致しています。この例では、オブジェクトが存在しなかったため、既存の値は空のリストです。また、提案された辞書と構成辞書はほとんど同じです(識別子はURLで使用されるため、最終的な要求では渡されません)。

APIC GUIでテナントに戻り、次の手順に従って存在を検証します。

- 「Contracts」 フォルダを展開します
- 「Standard」 フォルダを展開して、作成したコントラクトを表示します。

Name	Scope	QoS Class	Target DSCP	Subjects	Tags	Exported Tenants	Description
sql	VRF	Unspecified	Unspecified	sql			Contract Created Usin...
web	VRF	Unspecified	Unspecified	https			Contract Created Usin...

図1-53. ネットワーク構成を検証する

## ステップ 6 : ACI Ansible テナントアプリケーションプロファイルモジュールを確認する

EPGを使用してアプリケーションプロファイルを作成するには、次の四つのモジュールが必要です。

- aci\_ap
  - aci\_epg
  - aci\_epg\_to\_domain
  - aci\_epg\_to\_contract
1. 04\_aci\_tenant\_app\_pb.ymlファイルを開いて、各モジュールについて説明し、Playbookで提供されている例を確認します。

### アプリプロファイルモジュール

アプリケーションプロファイルの管理は、aci\_apモジュールを使用して行われます。ACIでは、アプリケーションプロファイルは、アプリケーションを構成するサービスのコンテナとして使用されます。それらはより組織的であるapため、新しいパラメータは一つだけです。これは、アプリケーションプロファイルが目的の状態にあることを確認するために使用されます。

#### 2. aci\_ap Playbookでタスクを確認する

Playbookの最初のタスク(8行目から17行目)は、EPGのアプリプロファイルが存在することを確認するために使用されます。

```
- name: ENSURE APPLICATION EXISTS
  aci_ap:
    host: "{{ inventory_hostname }}"
    username: "{{ username }}"
    password: "{{ password }}"
    state: "present"
    validate_certs: false
    tenant: "{{ tenant }}"
    ap: "{{ ap }}"
    descr: "App Profile Created Using Ansible"
```

### EPGモジュール

このaci\_epgモジュールは、アプリケーションのエンドポイントグループを管理するために使用されます。EPGにはいくつかの構成項目があり、モジュールがサポートするいくつかの項目を強調しています。

- **ap:** EPGが属するアプリケーションプロファイル名
- **epg:** EPGの名前
- **bd:** EPGに属するホストが関連付けられているブリッジドメイン。これにより、ホストが接続されるブロードキャストドメインと、ホストが使用できるIPサブネットが決まります。

#### 3. aci\_epgPlaybookでタスクを確認する

Playbookの2番目のタスク(19行目から31行目)は、アプリケーションに「web」および「db」EPGが存在することを確認するために使用されます。

```
- name: ENSURE APPLICATION EPGS EXISTS
  aci_epg:
    host: "{{ inventory_hostname }}"
    username: "{{ username }}"
    password: "{{ password }}"
    state: "present"
    validate_certs: False
    tenant: "{{ tenant }}"
    ap: "{{ ap }}"
    epg: "{{ item.epg }}"
    bd: "prod_bd"
    description: "EPG Created Using Ansible"
  with_items: "{{ epgs }}
```

このタスクを見ると、with\_itemsがepgsという名前の変数を使用していることがわかります。

- ./vars/intranet\_vars.ymlを開くと、Playbookの変数を含むファイルが表示されます。
- tenant値を、このラボで使用しているテナントの名前に変更します。

```
---
  tenant: CHANGEME
  ap: "intranet"
```

```

epgs:
  - epg: "web"
    encap: "21"
    contract_type: "both"
    consumer: "sql"
    provider: "web"
  - epg: "sql"
    encap: "22"
    contract_type: "provider"
    provider: "sql"
epg_contracts:
  - epg: "web"
    contract: "web"
    contract_type: "provider"
  - epg: "web"
    contract: "sql"
    contract_type: "consumer"
  - epg: "sql"
    contract: "sql"
    contract_type: "provider"

```

epgsキーは、前のPlaybookで使用されているリストに似ていますが、今、私たちはタスクごとに書き換えることなく、複数のタスクに同じデータを渡すことができます。

## EPGとドメインバインディング

このaci\_epg\_to\_domainモジュールは、EPGからドメインへのバインディング(VMMおよび物理)を管理するために使用されます。Playbookの次のタスク(33行目から48行目)

- **ap:** EPGが属するアプリケーションプロファイル名
- **epg:** ドメインに関連付けるEPGの名前
- **domain:** EPGに関連付けるVMMまたは物理ドメインの名前
- **domain\_type:** ドメインが物理であるか仮想であるかを判別します。オプションは、物理の場合は「phys」、ハイパーバイザー統合の場合は「vmm」です。
- **vm\_provider:** VMMドメインを管理する場合、これにより、ハイパーバイザーに使用されているベンダーが決まります。これは、domain\_typeがvmmの場合に必要です。
- **encap\_mode:** VMMドメインを管理する場合、これにより、VLANとVXLANのどちらを使用するかが決まります。
- **encap:** 静的割り当てを使用する場合、VLAN番号をVMMドメインに割り当てます。

```

- name: ENSURE DOMAIN IS BOUND TO EPG
  aci_epg_to_domain:
    host: "{{ inventory_hostname }}"
    username: "{{ username }}"
    password: "{{ password }}"
    state: "present"
    validate_certs: false
    tenant: "{{ tenant }}"
    ap: "{{ ap }}"
    epg: "{{ item.epg }}"
    domain: "aci_ansible_lab"
    domain_type: "vmm"
    vm_provider: "vmware"
    encap_mode: "auto"
    encap: "{{ item.encap }}"
    with_items: "{{ epes }}"

```

このタスクもepgs前のタスクと同じ変数を使用することに注意してください。これにより、構成データを2回書き直す必要がなくなり、タスク間で値の一貫性を保つことができます。

## EPGをContract[コントラクト]にバインドする

aci\_epg\_to\_contractモジュールは、EPGに関連付けられているコントラクトを管理するために使用されます。このモジュールは、aci\_tenant、aci\_ap、aci\_epg、およびaci\_contractモジュールの命名パラメータを使用します。追加のパラメータとして、EPGがコントラクトのであるproviderかconsumerコントラクトであるかを示すために使用されるcontract\_typeがあります。Playbookの次のタスク(50行目から62行目)

### 4. aci\_epg\_to\_contract Playbookでタスクを確認する

```
- name: ENSURE EPG IS ASSOCIATED TO CONTRACTS
```

```

aci_epg_to_contract:
  host: "{{ inventory_hostname }}"
  username: "{{ username }}"
  password: "{{ password }}"
  state: "present"
  validate_certs: False
  tenant: "{{ tenant }}"
  ap: "{{ ap }}"
  epg: "{{ item.epg }}"
  contract: "{{ item.contract }}"
  contract_type: "{{ item.contract_type }}"
  with_items: "{{ epg_contracts }}"

```

## ステップ7 : ACI Ansible テナントアプリケーションポリシーモジュールを実行する

多くの変数を持つPlaybookの場合、各変数を個別に渡すよりも、変数を含むファイルを渡す方が簡単です。Ansibleは@シンボルを使用して、Playbookに渡される変数がファイルであることを識別します。`--extra-vars "var=value"`の代わりに、`--extra-vars "@/path/to/file"`を使用します。テンプレートvarsファイルを作成すると、Playbookがどの変数を期待するかを覚えておく必要がないため、Playbookの再利用が容易になります。また、varsファイルは履歴目的で構成ファイルとして保持することもできます。

コマンドラインで次のコマンドを入力します。

```
ansible-playbook -i inventory 04_aci_tenant_app_pb.yml --extra-vars "@./vars/intranet_vars.yml"
```

```

$ ansible-playbook -i inventory 04_aci_tenant_app_pb.yml --extra-vars
"@./vars/intranet_vars.yml"
PLAY [ENSURE APPLICATION CONFIGURATION EXISTS] ****
*****
TASK [ENSURE APPLICATION EXISTS] ****
changed: [apic1.dcloud.cisco.com]

TASK [ENSURE APPLICATION EPGS EXISTS] ****
changed: [apic1.dcloud.cisco.com] => (item={'contract_type': u'both', u'epg': u'web', u'consumer': u'sql', u'encap': u'21', u'provider': u'web'})
changed: [apic1.dcloud.cisco.com] => (item={'contract_type': u'provider', u'epg': u'sql', u'encap': u'22', u'provider': u'sql'})

TASK [ENSURE DOMAIN IS BOUND TO EPG] ****
changed: [apic1.dcloud.cisco.com] => (item={'contract_type': u'both', u'epg': u'web', u'consumer': u'sql', u'encap': u'21', u'provider': u'web'})
changed: [apic1.dcloud.cisco.com] => (item={'contract_type': u'provider', u'epg': u'sql', u'encap': u'22', u'provider': u'sql'})

TASK [ENSURE EPG IS ASSOCIATED TO CONTRACTS] ****
changed: [apic1.dcloud.cisco.com] => (item={'contract_type': u'provider', u'epg': u'web', u'contract': u'web'})
changed: [apic1.dcloud.cisco.com] => (item={'contract_type': u'consumer', u'epg': u'web', u'contract': u'sql'})
changed: [apic1.dcloud.cisco.com] => (item={'contract_type': u'provider', u'epg': u'sql', u'contract': u'sql'})

PLAY RECAP ****
apic1.dcloud.cisco.com    : ok=4      changed=4      unreachable=0      failed=0

```

\$

APIC GUIでテナントに戻り、次の手順に従って存在を検証します。

- Application Profiles[アプリケーションプロファイル]フォルダを展開します
- Application EPGs[アプリケーションEPG]フォルダを展開して、作成したEPGを表示します。

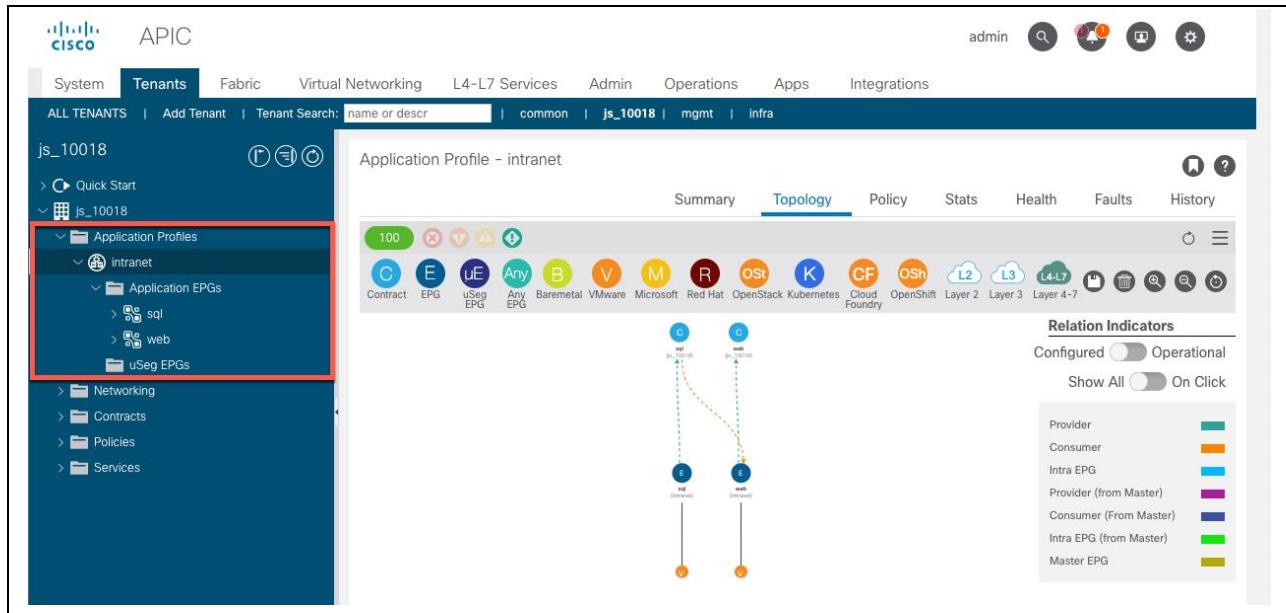


図1-54. ネットワーク構成を検証する

### お疲れ様でした！

すばらしい作業です。次のラボでは、すべてを1つのPlaybookにまとめる方法と、モジュールが利用できない場合の処理方法について説明します。

## 4.3. Ansible で ACI as Code としてまとめる

### 概要

Cisco Application Centric Infrastructure(ACI)には、APIを使用してプログラムでACIファブリックを構築および運用するのに役立つツールがいくつかあります。これらのツールには、APIインスペクター、ACI Cobra SDK、ACIツールキットなどがあります。これらのツールはすべて、他のDevNet Learning Labsでカバーされています。

この学習モジュールのラボでは、Ansibleを使用してACIを自動化する最新の方法を使用して説明します。Ansible、特にAnsibleのACIモジュールを使用する利点は、カスタムスクリプトを作成および開発する必要がなくなることです。Ansibleモジュールは基本的に、特定のタスクを実行するために作成されたスクリプトです。これにより、APIリクエストの調査、適切なAPIリクエストを行うためのコードの開発、コードのテストとデバッグにかかる時間を節約できます。

### 目的

- アプリケーションポリシー全体を管理するための単一のPlaybookを作成する
- 汎用ACIモジュールの使用
- ACIテナントブリッジドメインサブネットを作成する
- ACIテナントフィルターとContract[コントラクト]を作成する
- ACIテナントアプリケーションプロファイルを作成する
- ACIテナントエンドポイントグループを作成する

### ステップ 1：アプリケーションポリシー用の単一の Playbook を作成する

以前のラボでは、ACIでアプリケーションを個別のPlaybookに構築するために必要なさまざまな構成要素を分割していました。日常業務の場合、新しいアプリケーションを構成するプロセスを自動化するために使用できる単一のPlaybookがあると便利です。

注：続行する前に、コードサンプルディレクトリaci\_ansible\_learning\_labs\_code\_samples/intro\_moduleにいることを確認し、インストールしたPython仮想環境でAnsibleがアクティビ化されていることを確認してください。

### ステップ 2：Ansible タグと変数を確認する

#### タグ

シナリオ05\_aci\_deploy\_app.ymlは、いくつかの変更がファイルをvarsからのすべての変数が取得されるように作られて、このラボ内のすべてのタスクのコレクションです。コマンドシェルで、--list-tasks引数を使用してこのPlaybookのプレビューを生成します。

コマンドラインで次のコマンドを入力します：

```
ansible-playbook 05_aci_deploy_app.yml -i inventory --list-tasks
$ ansible-playbook -i inventory 05_aci_deploy_app.yml --list-tasks
playbook: 05_aci_deploy_app.yml

play #1 (apic): ENSURE APPLICATION CONFIGURATION EXISTS    TAGS: []
  tasks:
    TASK 01 - ENSURE APPLICATIONS TENANT EXISTS    TAGS: [app, bd, contract, epg, filter, tenant, vrf]
      TASK 02 - ENSURE TENANT VRF EXISTS    TAGS: [bd, vrf]
      TASK 03 - ENSURE TENANT BRIDGE DOMAINS AND EXIST    TAGS: [bd]
      TASK 04 - ENSURE BRIDGE DOMAINS HAVE SUBNETS    TAGS: []
      TASK 05 - ENSURE TENANT FILTERS EXIST    TAGS: [contract, filter]
      TASK 06 - ENSURE FILTERS HAVE FILTER ENTRIES    TAGS: [contract, filter]
      TASK 07 - ENSURE TENANT CONTRACTS EXIST    TAGS: [contract]
      TASK 08 - ENSURE CONTRACTS HAVE CONTRACT SUBJECTS    TAGS: [contract]
      TASK 09 - ENSURE CONTRACT SUBJECTS HAVE FILTERS    TAGS: [contract]
      TASK 10 - ENSURE APPLICATION EXISTS    TAGS: [app, epg]
      TASK 11 - ENSURE APPLICATION EPGS EXISTS    TAGS: [epg]
      TASK 12 - ENSURE DOMAIN IS BOUND TO EPG    TAGS: [epg]
      TASK 13 - ENSURE EPGS HAVE CONTRACTS    TAGS: [epg]
```

出力は、Playbookに13のタスクで一つのプレイがあることを示しています。各タスクは、アプリケーショ

ンに必要な構成が存在することを確認します。各タスクの最後にあるTAGSは、Playbook内のサブセットを実行する方法です。タグが使用されたときにタスクのすべての依存関係が実行されるように、タスクにタグを提供しました。たとえば、このPlaybookのすべてのタスクにはテナントオブジェクトが存在する必要があるため、テナントが存在することを確認するタスクには、プレイで使用されるすべてのタグが含まれます。

## Varsファイル

vars/intranet\_vars\_full\_config.ymlを開いて、アプリケーションの構成変数を表示します。

注：tenantの値CHANGEMEを、作成して使用しているテナントの名前に変更します。

```
---
tenant: CHANGEME
vrf: "prod_vrf"
bridge_domains:
  - bd: "prod_bd"
    gateway: "10.1.100.1"
    mask: "24"
    scope: "public"
ap: "intranet"
epgs:
  - epg: "web"
    bd: "prod_bd"
    encap: "21"
  - epg: "db"
    bd: "prod_bd"
    encap: "22"
epg_contracts:
  - epg: "web"
    contract: "web"
    contract_type: "provider"
  - epg: "web"
    contract: "sql"
    contract_type: "consumer"
  - epg: "db"
    contract: "sql"
    contract_type: "provider"
contracts:
  - contract: "web"
    subject: "https"
    filter: "https"
  - contract: "sql"
    subject: "sql"
    filter: "sql"
filters:
  - filter: "https"
    entry: "https"
    protocol: "tcp"
    port: "443"
  - filter: "sql"
    entry: "sql"
    protocol: "tcp"
    port: "1433"
```

このファイルは、前のPlaybookで使用したvarsファイルに似ていますが、このファイルには完全なアプリケーション構成があります。このファイルは、ハンドブックが期待するものの標準形式であるため、このファイルをテンプレートとして使用して、将来のすべてのアプリケーションの構成を構築できます。

## ステップ3：タグを使用して Playbook を実行する

ACI Ansibleモジュールはべき等です。つまり、モジュールは、存在する構成と要求されている構成の間に実際に違いがある場合にのみ変更を行います。これを示すために、このPlaybookを実行して、このvarsファイルを渡すことができます。ブリッジドメインに関連して再生するタスクを制限するには、--tags引数(または-t略して)を使用します。

コマンドラインで次のコマンドを入力します：

```
ansible-playbook -i inventory 05_aci_deploy_app.yml --extra-vars "@./vars/intranet_vars_full_config.yml" --tags bd
```

```
$ ansible-playbook 05_aci_deploy_app.yml -i inventory --extra-vars
```

```

"@./vars/intranet_vars_full_config.yml" --tags bd

PLAY [ENSURE APPLICATION CONFIGURATION EXISTS] ****
TASK [TASK 01 - ENSURE APPLICATIONS TENANT EXISTS] ****
ok: [apic1.dcloud.cisco.com]

TASK [TASK 02 - ENSURE TENANT VRF EXISTS] ****
ok: [apic1.dcloud.cisco.com]

TASK [TASK 03 - ENSURE TENANT BRIDGE DOMAINS AND EXIST] ****
ok: [apic1.dcloud.cisco.com] => (item={u'bd': u'prod_bd', u'scope': u'public', u'mask': u'24', u'gateway': u'10.1.100.1'})

PLAY RECAP ****
apic1.dcloud.cisco.com : ok=3    changed=0    unreachable=0    failed=0
$
```

このPlaybookにbdタグを使用すると、最初の三つのタスクだけが実行されます。最初の二つはBridgeDomainオブジェクトの依存関係であるため、bdタグも含まれています。

また、今回は変更された値が0に等しいことに注意してください。これは、モジュールがべき等であり、これらの構成がこのラボの前半で適用されたため、実際の変更がないためです。

## ステップ 4 : ACI REST API コマンドを実行する

Ansible ACIライブラリは、ACIで最も一般的に管理されるオブジェクトをサポートするモジュールを提供します。このaci\_restモジュールは、現在のモジュールが特定の機能をサポートしていないギャップを埋めるために作成されました。このモジュールを使用してAPICにAPIリクエストを行う方法を示します。

### ACI RESTモジュール

このaci\_restモジュールは、XMLまたはJSONのいずれかを使用してAPICに対してGET、POST、またはDELETE要求を行う方法を提供します。このモジュールには、受け入れるパラメータがいくつかあります。

- method: リクエストのHTTPメソッド(GET、POST、またはDELETE)
- path: APIリクエストのURL
- src: ファイルを使用してPOSTリクエストの本文を提供するために使用できます
- content: タスクでPOSTリクエストの本文を提供するために使用できます

06\_aci\_rest\_pb.ymlを開き、pathパラメータの**CHANGEME**を、このラボで作業しているテナントの値に変更します。

```

---
- name: USE ACI REST MODULE
  hosts: apic
  connection: local
  gather_facts: False

  tasks:
    - name: ENSURE TENANT HAS L3 EXTERNAL NETWORK
      aci_rest:
        host: "{{ inventory_hostname }}"
        username: "{{ username }}"
        password: "{{ password }}"
        validate_certs: False
        method: "post"
        path: "api/mo/uni/tn-CHANGEME/out-corp_13.json"
        content: {"l3extOut": {"attributes": {"descr": "Created Using Ansible", "name": "corp_13"}}}
```

このPlaybookは、レイヤ3外部ネットワークをテナントに追加するために使用されます。この要求は非常に基本的であり、外部ネットワークのシェルを作成するだけですが、モジュール自体を使用すると、要求に必要な数の構成パラメータを入れることができます。モジュールは正確性のチェックを実行しないため、URLまたはコンテンツが有効でない場合、リクエストは失敗します。

コマンドラインで次のコマンドを入力し、-vvvをもう一度使用して、テナント名が使用されたことを確認します。

```

ansible-playbook -i inventory 06_aci_rest_pb.yml -vvv
$ ansible-playbook -i inventory 06_aci_rest_pb.yml -vvv

ok: [sandboxapicdc.cisco.com] => {
    "changed": false,
    "error_code": 0,
    "error_text": "Success",
    "failed": false,
    "imdata": [],
    "invocation": {
        "module_args": {
            "content": "{\"l3extOut\": {\"attributes\": {\"descr\": \"Created Using Ansible\", \"name\": \"corp_l3\"}}}",
            "host": "apic1.dccloud.cisco.com",
            "hostname": "apic1.dccloud.cisco.com",
            "method": "post",
            "password": "VALUE_SPECIFIED_IN_NO_LOG_PARAMETER",
            "path": "api/mo/uni/tn-jt_1234/out-corp_l3.json",
            "protocol": "https",
            "src": null,
            "timeout": 30,
            "use_proxy": true,
            "use_ssl": true,
            "username": "admin",
            "validate_certs": false
        }
    },
    "response": "OK (30 bytes)",
    "status": 200,
    "totalCount": "0"
}
$ 

```

APIC GUIでテナントに戻り、次の手順に従って存在を検証します。

- Networking[ネットワーク]フォルダを展開します
- External Routed Networks[外部ルーティングネットワーク]フォルダを展開して、corp\_l3ネットワークが作成されたことを確認します

Name	Description
corp_l3	Created Using Ansible

図1-55. ネットワーク構成を検証する

## ステップ 5：ラボのクリーンアップを実行する

このラボの構成をクリーンアップするには、aci\_restモジュールを使用してテナントのDELETEリクエストを作成します。

07\_lab\_cleanup.yml playbookを開きpathパラメータのCHANGEMEを編集して、このラボで作業しているテ

ナントの値に変更します。

```
---
- name: USE ACI REST MODULE
  hosts: apic
  connection: local
  gather_facts: False

  tasks:
    - name: ENSURE LAB WORK IS CLEANED UP
      aci_rest:
        host: "{{ inventory_hostname }}"
        username: "{{ username }}"
        password: "{{ password }}"
        method: "delete"
        validate_certs: False
        path: "api/mo/uni/tn-CHANGEME.json"
```

このPlaybookを実行すると、テナントとすべての構成がラボAPICから削除されます。

コマンドラインで次のコマンドを入力します。

```
ansible-playbook -i inventory 07_lab_cleanup.yml
```

```
$ ansible-playbook -i inventory 07_lab_cleanup.yml
```

```
PLAY [USE ACI REST MODULE] ****
TASK [ENSURE LAB WORK IS CLEANED UP] ****
changed: [apic1.dcloud.cisco.com]

PLAY RECAP ****
sandboxapicdc.cisco.com    : ok=1    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
$
```

**お疲れ様でした！**

素晴らしい仕事です！これで、ACI Ansibleモジュールを使用してACI環境全体をas Code[コードとして]管理する方法をしっかりと理解できました。

## 5. ACI と Terraform の紹介

HashiCorp Terraformを使用したCisco Application Centric Infrastructure(ACI)の自動化とプログラマビリティの概要

### 「Terraformを利用したCisco ACIの自動化の開始」

ACIでTerraformを使用する方法と、インフラストラクチャをコードとして使用してテナントを作成する方法について説明します

### 「Terraformを用いたアプリケーションとネットワークポリシー」

Terraformリソースを使用してACIアプリケーションとネットワークポリシーを制御します。

### 「ACI REST APIを汎用ACI Terraformリソースでラップする」

一般的なACI Terraformリソースの使用方法について学ぶ

## 5.1. Terraform で Cisco ACI の自動化を実行

Cisco Application Centric Infrastructure(ACI)には、APIを使用してプログラムでACIファブリックを構築および運用するのに役立つツールがいくつかあります。これらのツールには、APIインスペクター、ACI Cobra SDK、ACI Toolkit、Ansibleなどがあります。これらのツールはすべて、他のDevNet LearningLabsでカバーされています。

この学習モジュールのラボでは、Terraformを使用してACIを自動化する最新の方法を使用して説明します。Terraform、特にACI用のTerraformリソースとデータソースを使用する利点は、カスタムスクリプトを作成および開発する必要がなくなることです。Terraformリソースは、基本的に、特定のタスクを実行する事前に作成されたコードのブロックです。これにより、APIリクエストの調査、適切なAPIリクエストを作成するためのコードの開発、コードのテストとデバッグにかかる時間を節約できます。

### 目的

- TerraformおよびACI環境をセットアップします
- ACI TerraformProviderとそのリソースおよびACIのデータソースについて理解する
- Terraformを使用してACIテナントを作成する

### ステップ 1: ACI Terraform プロバイダの概要

Terraformをインストールしたので、Terraformプロバイダとは何かを理解しましょう。

Terraformプロバイダは、Terraformバイナリがサードパーティシステムと対話できるようにするリソースとデータソースのセットです。リソースは、インフラストラクチャのユニットを追加、変更、または破棄するために使用されます。データソースは、インフラストラクチャの既存のユニットを表すために使用され、他のリソースまたはデータソースでそれを補間または参照できるようにします。ACI Terraformプロバイダの場合、これらのリソースとデータソースは、アプリケーションポリシーインフラストラクチャコントローラーまたはAPICのACI構成要素を表すために使用されます。

ACI TerraformプロバイダのようなTerraformプロバイダの使用を開始する最初のステップは、Terraformプランでプロバイダを定義することです。Terraform Planは、HashiCorp構成言語(HCL)を使用してインフラストラクチャを表すプロバイダ、リソース、およびデータソースを記述する構成ファイルです。デフォルトのTerraformPlanファイルの名前はmain.tfです。

このラボでは、このラボの最初に複製されたリポジトリに付属のmain.tfファイルを使用しています。main.tfファイルはディレクトリaci\_terraform\_learning\_labs\_code\_samples/intro\_moduleにあります。

```
terraform {
  required_providers {
    aci = {
      source = "CiscoDevNet/aci"
    }
  }
}

# Configure the provider with your Cisco APIC credentials.
provider "aci" {
  # APIC Username
  username = var.user.username
  # APIC Password
  password = var.user.password
  # APIC URL
  url      = var.user.url
  insecure = true
}

...

```

main.tfファイルの最初の部分がプロバイダのソースを指し、次にACIプロバイダの定義とそれをインスタンス化するために必要なパラメータが含まれていることがわかります。

ACIのプロバイダ構成のこの例では、username、passwordおよびURLを、APICのAPI接続を確立するために使用されます。

これらのパラメータは、ACIプロバイダによって使用され、各リソースをインスタンス化するか、各データソースの情報を取得します。複数のプロバイダを定義main.tfして、同じプラン内の複数のサードパーティシステムと対話できるようにすることができます。

variable.tfファイルで定義されている変数を示していることvar.user.username、var.user.password、var.user.urlに注意してください。

```

variable "user" {
  description = "Login information"
  type        = map
  default     = {
    username = "admin"
    password = "ciscopsdt"
    url      = "https://sandboxapicdc.cisco.com"
  }
}

```

変数は、APICサンドボックスの公的に解決可能なDNS名であるsandboxapicdc.cisco.comを指します。

今回の環境に合わせて2か所を変更します。

```

variable "user" {
  description = "Login information"
  type        = map
  default     = {
    username = "admin"
    password = "!3G@!4@Y"
    url      = "https://sandboxapicdc.cisco.com"
  }
}

```

#### 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プランの初期化

上に示しましたmain.tfを含む、aci\_terraform\_learning\_labs\_code\_samples/intro\_moduleディレクトリにいることを確認してください。

計画を初期化するには、次のコマンドを実行します。

```
$ terraform init
```

```
Initializing the backend...
```

```
Initializing provider plugins...
```

```
- Finding latest version of ciscodenvnet/aci...
- Installing ciscodenvnet/aci v0.7.1...
- Installed ciscodenvnet/aci v0.7.1 (signed by a HashiCorp partner, key ID
433649E2C56309DE)
```

Partner and community providers are signed by their developers.

If you'd like to know more about provider signing, you can read about it here:  
<https://www.terraform.io/docs/plugins/signing.html>.

Terraform has created a lock file .terraform.lock.hcl to record the provider selections it made above. Include this file in your version control repository so that Terraform can guarantee to make the same selections by default when you run "terraform init" in the future.

**Terraform has been successfully initialized!**

You may now begin working with Terraform. Try running "terraform plan" to see any changes that are required for your infrastructure. All Terraform commands should now work.

If you ever set or change modules or backend configuration for Terraform, rerun this command to reinitialize your working directory. If you forget, other commands will detect it and remind you to do so if necessary.

これで、Terraformの使用を開始できます。次のステップでterraform planを実行して(ここではまだ実行しません)、インフラストラクチャに必要な変更を確認してください。これで、すべてのTerraformコマンドが機能するはずです。

Terraformのモジュールまたはバックエンド構成を設定または変更したことがある場合は、このコマンドを再実行して、作業ディレクトリを再初期化します。忘れた場合、他のコマンドがそれを検出し、必要に応じてそうするように通知します。

これでプランが初期化され、terraform.terraformが現在のフォルダにディレクトリを作成しました。

```
$ ls -a
.          ..      .terraform  main.tf    variable.tf
$
```

このディレクトリには、ACITerraformプロバイダのバイナリ実行可能ファイルが含まれます。

#### ACIプロバイダーパラメーター

この例では、ACIプロバイダを構成するために使用可能なパラメータのサブセットを示しました。

構成オプションの完全なリストは、ドキュメントに記載されています。

### ステップ 2: Terraform プランで ACI リソースを確認する

ACIテナントリソースは幅広いACIテナント構成をサポートしますが、テナントネットワーク、ポリシー、およびEPGの管理に必要なコアリソースを紹介します。

このラボでは、各リソースの目的、固有のパラメータ、および期待される出力について説明します。このモジュールの最後に、テナント構成が存在することを保証するTerraformプランを作成して使用します。以下のACIツールキットの図は、このTerraformプランが構成し、存在することを確認する構成です。

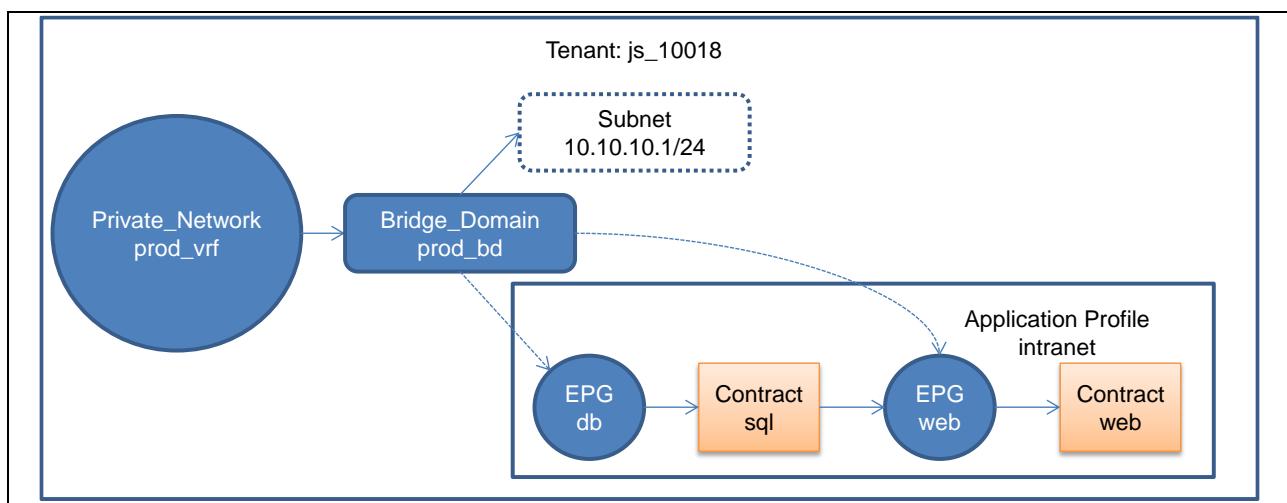


図1-56. Terraformの例

#### ACI Terraformプランの表示

ディレクトリaci\_terraform\_learning\_labs\_code\_samples/intro\_moduleにあるmain.tfファイルを確認します。

```
terraform {
  required_providers {
    aci = {
      source = "CiscoDevNet/aci"
    }
  }
}

# Configure the provider with your Cisco APIC credentials.
provider "aci" {
  # APIC Username
  username = var.user.username
  # APIC Password
  password = var.user.password
  # APIC URL
  url      = var.user.url
  insecure = true
}
```

```

}

# Define an ACI Tenant Resource.
resource "aci_tenant" "terraform_tenant" {
    name      = "terraform_tenant"
    description = "This tenant is created by terraform"
}

```

前のセクションで、プロバイダのソースとプロバイダの定義についてはすでに説明しました。ここでは、リソースの定義に集中します。

## テナントリソースの使用

このaci\_tenantリソースは、APICでテナントを管理するために使用されます。リソース名(aci\_tenantこの場合)の横には、リソースterraform\_tenantのこのインスタンスの表現を含む変数の名前があります。

このリソースには、現時点で知つておく必要のある2つのパラメータがあります。

- **name** : これは、APICから構成を管理または取得するテナントの名前です。
- **description** : これは、このテナントが何に使用されるかを説明できるテナントの説明です。

注：テナントはすべてのテナント関連オブジェクトのルート親であるため、このリソース変数のid属性は、このテナントの別のリソース部分で使用されます。

このラボの最初に複製したディレクトリaci\_terraform\_learning\_labs\_code\_samples/intro\_module内のmain.tf内のファイルを開きます。このプランの最初で唯一のリソースは、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されます。

このリソースは、apic1.dcloud.cisco.comにリンクするプロバイダ定義で定義されたAPICで作成されます。

テナントを一意にするにnameは、リソースのパラメータを、次の例のように、イニシャルの後に電話番号を続けるなどの一意の文字列に変更します。

```

# Define an ACI Tenant Resource.
resource "aci_tenant" "terraform_tenant" {
    name      = "jt_1234"
    description = "This tenant is created by terraform"
}

```

## ステップ 3: Terraform プランコマンドを実行します

次のコマンドを実行します。

```

$ terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.
-----
```

```

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:

```

# aci_tenant.terraform_tenant will be created
+ resource "aci_tenant" "terraform_tenant" {
    + annotation = "orchestrator:terraform"
    + description = "This tenant is created by terraform"
    + id          = (known after apply)
    + name        = "jt_1234"
    + name_alias  = (known after apply)
}
```

Plan: 1 to add, 0 to change, 0 to destroy.

```

-----
```

Note: You didn't specify an "-out" parameter to save this plan, so Terraform
can't guarantee that exactly these actions will be performed if

"terraform apply" is subsequently run.

Terraformは、この計画を適用するときに実行されるアクションの詳細を示していることがわかります。この場合、このプランはjt\_1234と呼ばれる新しいテナントを作成します。あなたの場合、テナント名にはあなたのイニシャルと電話番号が含まれます。

## ステップ 4 : ACI Terraform プランを適用し、テナントを作成する

次に、プランを適用します。プランの一部は、APICでテナントを作成することです。実行する前に計画を確認する必要があります。

### ACITerraformプランを適用する

次のコマンドを実行します。

```
$ terraform apply
```

```
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:

```
# aci_tenant.terraform_tenant will be created  
+ resource "aci_tenant" "terraform_tenant" {  
    + annotation = "orchestrator:terraform"  
    + description = "This tenant is created by terraform"  
    + id          = (known after apply)  
    + name        = "jt_1234"  
    + name_alias  = (known after apply)  
}
```

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?

```
Terraform will perform the actions described above.  
Only 'yes' will be accepted to approve.
```

Enter a value:

terraform applyは、terraform planの出力を表示し、yesと、Enterキーを押して確認するように求めます。

注: テナントはすべてのテナント関連オブジェクトのルート親であるためid、このリソース変数のid属性は、このテナントの別のリソース部分で使用されます。

Enter a value: yes

```
aci_tenant.terraform_tenant: Creating...  
aci_tenant.terraform_tenant: Creation complete after 3s [id=uni/tn-jt_1234]
```

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

結果は、テナント名を含むID(ACI DNに相当)を持つ新しいテナントを作成したことを示しています。

また、Terraformがディレクトリに状態ファイルterraform.tfstateを作成したこともわかります。

```
$ ls -l  
main.tf           terraform.tfstate variable.tf
```

APICをチェックして、新しいテナントが存在することを検証することもできます。

- Tenantsをクリックします
- 自身で作成したテナントをダブルクリックして開きます

Name	Alias	Description	Bridge Domains	VRFs	EPGs	Health Score
common			1	2	0	<span style="background-color: green; color: white; border-radius: 50%; padding: 2px 5px;">100</span>
infra			2	2	2	<span style="background-color: green; color: white; border-radius: 50%; padding: 2px 5px;">100</span>
js_10018		Tenant Created Using A...	0	0	0	<span style="background-color: green; color: white; border-radius: 50%; padding: 2px 5px;">100</span>
mgmt			1	2	0	<span style="background-color: green; color: white; border-radius: 50%; padding: 2px 5px;">100</span>

図1-57. オープンテナント

テナントを開くと、左側のナビゲーションにその子が表示されます。これらを使用して、計画が他の Terraform ACI ラーニングラボ全体で期待どおりに機能したことを見証できます。

Domain	Count
S...	0
A...	0

図1-58. テナントの子供

### お疲れ様でした！

素晴らしい仕事。次のラボでは、Terraformを使用してアプリケーションとネットワークのポリシーを管理する方法を見ていきます。

## 5.2. Terraform で Application と Network Policy

### 概要

Cisco Application Centric Infrastructure(ACI)には、APIを使用してプログラムでACIファブリックを構築および運用するのに役立つツールがいくつかあります。これらのツールには、APIインスペクター、ACI Cobra SDK、ACI Toolkit、Ansibleなどがあります。これらのツールはすべて、他のDevNet LearningLabsでカバーされています。

この学習モジュールのラボでは、Terraformを使用してACIを自動化する最新の方法を使用して説明します。Terraform、特にACI用のTerraformリソースとデータソースを使用する利点は、カスタムスクリプトを作成および開発する必要がなくなることです。Terraformリソースは、基本的に、特定のタスクを実行する事前に作成されたコードのブロックです。これにより、APIリクエストの調査、適切なAPIリクエストを行うためのコードの開発、コードのテストとデバッグにかかる時間を節約できます。

### 目的

- ACIテナントVRFを作成する
- ACIテナントブリッジドメインを作成する
- ACIテナントブリッジドメインサブネットを作成する
- ACIテナントフィルターとコントラクトを作成する
- ACIテナントアプリケーションプロファイルを作成する
- ACIテナントエンドポイントグループを作成する

### ACIテナントネットワークリソースを確認する

テナントネットワーク構造に関連するTerraformリソースはいくつかありますが、アプリケーションにネットワーク接続を提供するために必要な3つに焦点を当てます。

- aci\_vrf
- aci\_bridge\_domain
- aci\_subnet

これらの各リソースについて説明し、Terraformを使用して3つのリソースすべてをプロビジョニングし、適切なネットワークオブジェクトが存在することを確認します。

注: 続行する前に、コードサンプルディレクトリaci\_terraform\_learning\_labs/network\_policiesにあり、PATHにterraformバイナリがあることを確認してください。

### ステップ 1: ACITerraform テナントネットワークリソースを確認する

#### VRFリソース

このaci\_vrfリソースは、APICでテナントVRFを管理するために使用されます。このリソースには、注意すべき3つの重要なパラメータがあります。

- **name:** 新しいVRFに付ける名前。
- **tenant\_dn:** このVRFの親テナントを参照するDN
- **pc\_enf\_pref:** VRFがセキュリティポリシーを実施するかどうかを決定します
- **pc\_enf\_dir:** ポリシーがスイッチからの入力または出力で適用されるかどうかを決定します。

構成オプションの完全なリストは、ドキュメントに記載されています。

リポジトリディレクトリのnetwork\_policiesaci\_terraform\_learning\_labs\_code\_samplesフォルダにあるmain.tfファイルを開きます。最初のリソースは、前のラボで使用したのと同じテナントです。コメントされたリソース# Define an ACI Tenant VRF Resourceを見てください。

```
# Define an ACI Tenant VRF Resource.
resource "aci_vrf" "terraform_vrf" {
    tenant_dn    = aci_tenant.terraform_tenant.id
    description  = "VRF Created Using Terraform"
    name         = var.vrf
}
```

APICによって提供されるデフォルト値を使用するだけなので、このリソース定義にはpc\_enfパラメータは含まれません。

このリソース定義では、vrf変数を使用します。これは、Terraform変数機能を使用して変数をプランに提供する方法を示します。このアプローチの利点は、別のVRFを使用するたびにファイルを手動で編集しなくて

も、プランを再利用できることです。

また、**interpolation**を使用して、テナントリソースインスタンス変数を使用してテナントDNを参照します。

次のタスクでは、関連するブリッジドメインを追加する方法を見ていきます。

## ブリッジドメインリソース

aci\_bdリソースは、ブリッジドメインを管理するために使用されます。このリソースにはいくつかのパラメータがあります。ここにもっと重要なことがあります：

- **name**: ブリッジドメインの名前。
- **tenant\_dn** このBDの親テナントを参照するDN。
- **relation\_fv\_rs\_ctx**: ブリッジドメインに関連付けられているサブネットが属するVRFのDN。

main.tfファイルで、コメントで示されたリソースを# Define an ACI Tenant BD Resourceから調べます

```
# Define an ACI Tenant BD Resource
resource "aci_bridge_domain" "terraform_bd" {
    tenant_dn      = aci_tenant.terraform_tenant.id
    relation_fv_rs_ctx = aci_vrf.terraform_vrf.id
    description     = "BD Created Using Terraform"
    name           = var.bd
}
```

このリソースは、**interpolation**を使用して、プランで以前に定義されたテナントとVRF DNを参照しています。これは、新しいVRFで作業していることを確認するために重要です。

nameパラメータに導入された新しい変数もあります。

## サブネットリソース

このaci\_subnetリソースは、ブリッジドメインに属するサブネットを管理するために使用されます。サブネットはブリッジドメインの子オブジェクトであるparent\_dnため、このモジュールではパラメータが使用されます。より一般的なパラメータは次のとおりです。

- **parent\_dn**: サブネットの親BDのDN。
- **ip**: サブネット上のホストのサブネットマスクを含むゲートウェイアドレス。
- **scope**: サブネットを外部にアドバタイズする(パブリック)か、VRFにプライベートに保つ(プライベート)か、他のテナントと共有する(共有)かを決定します。

ここで、静的に定義することを避けるためにparent\_dn、**interpolation**を使用してBDDNを参照します。

のデフォルト値scopeが**private**であることを指摘するのは重要です。VRFまたはACIファブリックの外部でネットワークに到達できるようにする場合は、リソースのスコープを**public**に明示的に設定する計画を作成する必要があります。

main.tfファイルで、次のリソースを調べます。

```
# Define an ACI Tenant BD Subnet Resource.
resource "aci_subnet" "terraform_bd_subnet" {
    parent_dn      = aci_bridge_domain.terraform_bd.id
    description     = "Subnet Created Using Terraform"
    ip             = var.subnet
    scope          = "public"
}
```

## ステップ 2: プランを適用し、テナントネットワークポリシーを作成します

Terraformプランのテナントネットワークリソースに精通したので、次はこのプランを適用します。

で、このモジュールでは最初のラボ、私たちは、テナントを作成しました。このプランを実行すると、テナントに新しいVRF、ブリッジドメイン、およびサブネットが作成されます。

Terraformを使用すると、プランで使用する変数をとvariable.tf呼ばれる別のファイルで定義できます。

variable.tfファイルで、新しいvrf、bdおよびsubnet変数を調べます。

```
variable "tenant" {
    type  = string
    default = "terraform-tenant"
}
```

```

variable "vrf" {
  type    = string
  default = "prod_vrf"
}
variable "bd" {
  type    = string
  default = "prod_bd"
}
variable "subnet" {
  type    = string
  default = "10.10.101.1/24"
}

```

テナントを一意にするには、tenant変数をイニシャルのような一意の文字列に変更し、その後に次のような電話番号を続けます。

```

variable "tenant" {
  type    = string
  default = "jt_1234"
}
variable "vrf" {
  type    = string
  default = "prod_vrf"
}
variable "bd" {
  type    = string
  default = "prod_bd"
}
variable "subnet" {
  type    = string
  default = "10.10.101.1/24"
}

```

前のセクションで説明した内容main.tfが含まれている

aci\_terraform\_learning\_labs\_code\_samples/network\_policiesディレクトリにいることを確認してください。

正しいディレクトリにいるときに、次のコマンドを実行します。

```
$ terraform init
```

「Terraform has been successfully initialized! (Terraformが正常に初期化されました!)」と表示されます。このコマンドを実行した後、他の情報とともにメッセージを表示します。

次に、次のapplyコマンドを実行できます。

```
$ terraform apply
```

```
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:

```
# aci_bridge_domain.terraform_bd will be created
+ resource "aci_bridge_domain" "terraform_bd" {
    + annotation          = "orchestrator:terraform"
    + arp_flood           = (known after apply)
    + bridge_domain_type = (known after apply)
    + description         = "BD Created Using Terraform"
    + ep_clear             = (known after apply)
    + ep_move_detect_mode = (known after apply)
    + host_based_routing   = (known after apply)
    + id                  = (known after apply)
    + intersite_bum_traffic_allow = (known after apply)
    + intersite_l2_stretch   = (known after apply)
    + ip_learning          = (known after apply)
    + ipv6_mcast_allow     = (known after apply)
    + limit_ip_learn_to_subnets = (known after apply)
    + ll_addr              = (known after apply)
    + mac                  = (known after apply)
    + mcast_allow          = (known after apply)
    + multi_dst_pkt_act    = (known after apply)
```

```

+ name                  = "prod_bd"
+ name_alias            = (known after apply)
+ optimize_wan_bandwidth = (known after apply)
+ relation_fv_rs_ctx    = (known after apply)
+ tenant_dn              = (known after apply)
+ unicast_route          = (known after apply)
+ unk_mac_unicast_act    = (known after apply)
+ unk_mcast_act          = (known after apply)
+ v6unk_mcast_act        = (known after apply)
+ vmac                  = (known after apply)
}

# aci_subnet.terraform_bd_subnet will be created
+ resource "aci_subnet" "terraform_bd_subnet" {
  + annotation = "orchestrator:terraform"
  + ctrl       = (known after apply)
  + description = "Subnet Created Using Terraform"
  + id         = (known after apply)
  + ip         = "10.10.101.1/24"
  + name_alias = (known after apply)
  + parent_dn   = (known after apply)
  + preferred   = (known after apply)
  + scope      = (known after apply)
  + virtual     = (known after apply)
}

# aci_tenant.terraform_tenant will be created
+ resource "aci_tenant" "terraform_tenant" {
  + annotation = "orchestrator:terraform"
  + description = "This tenant is created by terraform"
  + id         = (known after apply)
  + name       = "jt_1234"
  + name_alias = (known after apply)
}

# aci_vrf.terraform_vrf will be created
+ resource "aci_vrf" "terraform_vrf" {
  + annotation      = "orchestrator:terraform"
  + bd_enforced_enable = (known after apply)
  + description     = "VRF Created Using Terraform"
  + id              = (known after apply)
  + ip_data_plane_learning = (known after apply)
  + knw_mcast_act   = (known after apply)
  + name             = "prod_vrf"
  + name_alias       = (known after apply)
  + pc_enf_dir       = (known after apply)
  + pc_enf_pref      = (known after apply)
  + tenant_dn        = (known after apply)
}

```

Plan: 4 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?  
Terraform will perform the actions described above.  
Only 'yes' will be accepted to approve.

Enter a value:

terraform applyは、terraform planの出力を表示し、yesと入力してEnterキーを押して確認するように求めます。

Enter a value: **yes**

```

aci_tenant.terraform_tenant: Creating...
aci_tenant.terraform_tenant: Creation complete after 2s [id=uni/tn-jt_1234]
aci_vrf.terraform_vrf: Creating...
aci_vrf.terraform_vrf: Creation complete after 3s [id=uni/tn-jt_1234/ctx-prod_vrf]
aci_bridge_domain.terraform_bd: Creating...
aci_bridge_domain.terraform_bd: Creation complete after 4s
[id=uni/tn-jt_1234/BD-prod_bd]

```

```

aci_subnet.terraform_bd_subnet: Creating...
aci_subnet.terraform_bd_subnet: Creation complete after 1s
[ id=uni/tn-jt_1234/BD-prod_bd/subnet-[10.10.101.1/24] ]

```

```

Apply complete! Resources: 4 added, 0 changed, 0 destroyed.
$
```

APIC GUIでテナントに戻り、次の手順に従ってネットワークが存在することを確認します。

- 「Networking」 フォルダを展開します
- 「Bridge Domains」 フォルダを展開して、作成したprod\_dbブリッジドメインを表示します。
- 「prod\_db」 フォルダを展開します
- 「Subnets」 フォルダを展開して、作成したサブネットを表示します
- 「VRFs」 フォルダを展開して、作成したVRFを表示します

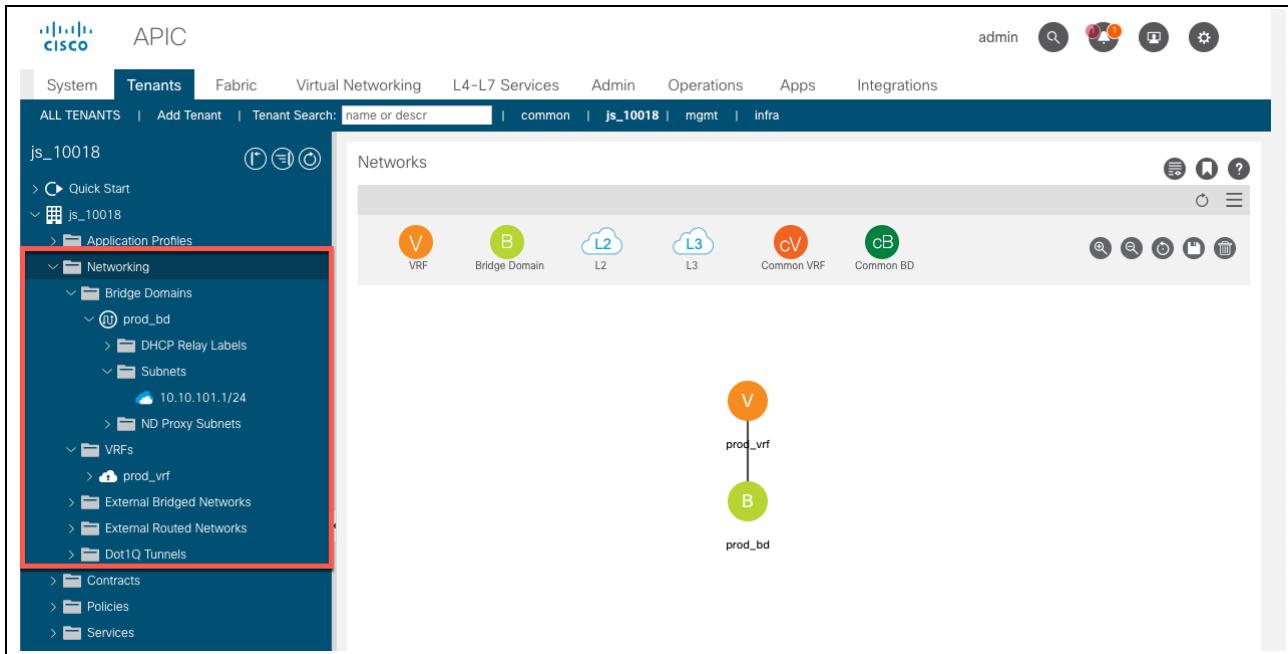


図1-59. ネットワーク構成を検証する

### ステップ 3: ACITerraform テナントポリシー/コントラクトリソースを確認する

アプリケーションのエンドポイントグループのポリシーを構築するために必要な4つのリソースは次のとおりです。

- aci\_filter
  - aci\_filter\_entry
  - aci\_contract
  - aci\_contract\_subject
1. ディレクトリをコードサンプルディレクトリに変更します。  
aci\_terraform\_learning\_labs\_code\_samples/contract\_policies
  2. このディレクトリを初期化するためにterraform initを実行します。
  3. main.tfファイルを開くと、各リソースについて説明し、計画で提供されている例を確認します。

#### フィルタおよびフィルタエントリリソース

このaci\_filterリソースは、APICでフィルタオブジェクトを管理するために使用され、次の属性があります。

- **name** : フィルタの名前
- **tenant\_dn** : このフィルタの親テナントを参照するDN

aci\_filter\_entryフィルタに関する付加情報のいくつかがされているネットワークプロトコルのエントリを管理するために使用されます。

- **name** : フィルタエントリの名前
- **filter\_dn** : 親フィルタを参照するDN
- **ether\_t** : オブジェクトのイーサネットタイプを設定します。「ip」が最も一般的なオプションです。
- **prot** : ether\_typeが「ip」の場合に使用可能で、オブジェクトのIPプロトコルを決定するために使用さ

れます。最も一般的なオプションは、「tcp」、「udp」、および「icmp」です。

- **d\_from\_port** : TCPおよびUDPフィルターエントリで使用可能で、宛先ポート範囲の開始ポートを設定するために使用されます
- **d\_to\_port** : TCPおよびUDPフィルターエントリで使用可能で、宛先ポート範囲の終了ポートを設定するために使用されます

#### 4. プランの最初のaci\_filterおよびaci\_filter\_entryリソースを表示します。

```
# Define an ACI Filter Resource.
resource "aci_filter" "terraform_filter" {
    for_each      = var.filters
    tenant_dn    = aci_tenant.terraform_tenant.id
    description  = "This is filter ${each.key} created by terraform"
    name         = each.value.filter
}

# Define an ACI Filter Entry Resource.
resource "aci_filter_entry" "terraform_filter_entry" {
    for_each      = var.filters
    filter_dn    = aci_filter.terraform_filter[each.key].id
    name         = each.value.entry
    ether_t      = "ipv4"
    prot         = each.value.protocol
    d_from_port  = each.value.port
    d_to_port    = each.value.port
}
```

これらのリソースは、と呼ばれるリソースメタ引数for\_eachを使用します。これにより、プロバイダはマップをループして複数のリソースを作成できます。アプリケーションに存在する必要のある2つのフィルタがあるため、最初のリソースはvar.filters、name引数に入力される2つの「フィルタ」を含むと呼ばれるマップを使用します。

2番目のリソースには、可変値を持つ複数の引数があるため、各フィールドにマッピングを提供して、どの引数が特定の値を取得するかを識別します。ACIは各TCP / UDPフィルターエントリをポートの範囲として扱いますが、エントリの範囲は1であるため、d\_from\_portとd\_to\_portとの両方に同じ値を使用できます。フィルタは最初のタスクのフィルタにマップされていることに注意してください。このタスクでは、これらの各フィルタに正しいポートフィルタエントリが割り当てられていることを確認します。

#### 5. ファイルvariable.tfから変数filtersを表示します。

```
variable "filters" {
    description = "Create filters with these names and ports"
    type        = map
    default     = {
        filter_https = {
            filter  = "https",
            entry   = "https",
            protocol = "tcp",
            port    = "443"
        },
        filter_sql = {
            filter  = "sql",
            entry   = "sql",
            protocol = "tcp",
            port    = "1433"
        }
    }
}
```

### コントラクトおよびコントラクト対象リソース

#### 6. プランのaci\_contractとaci\_contract\_subjectリソースを表示します。

aci\_contractリソースは、コントラクトを管理するために使用され、aci\_contract\_subjectリソースは、主コントラクトに関連付けられているかのフィルタを管理するために使用されます。

aci\_contractリソースで利用できるより重要なオプションの2つは次のとおりです。

- **name** : Contractオブジェクトの名前
- **scope** : コントラクトの範囲を決定します。オプションは、「context」、「application-profile」、「tenant」、および「global」です。コントラクトのスコープは、関連するEPGによってコントラクトがどのように提供および消費されるかを決定し、コントラクトを効率的に使用する方法です。

「context」は、コントラクト範囲のVRFと同じです

name引数に加えて、aci\_contract\_subjectリソースには次の2つの引数があります。

- **relation\_vz\_rs\_subj\_filt\_att** : フィルタとサブジェクト間の関係を作成するために使用するフィルタのDNのリスト。
- **rev\_fit\_ports** : コントラクトはステートレスであるため、ACIファブリックがポートを反転して元のホストに戻るトラフィックを許可するかどうかを決定します。この議論は計画には示されていません。

```
# Define an ACI Contract Resource.
resource "aci_contract" "terraform_contract" {
  for_each      = var.contracts
  tenant_dn    = aci_tenant.terraform_tenant.id
  name         = each.value.contract
  description   = "Contract created using Terraform"
  scope        = "context"
}

# Define an ACI Contract Subject Resource.
resource "aci_contract_subject" "terraform_contract_subject" {
  for_each      = var.contracts
  contract_dn  = aci_contract.terraform_contract[each.key].id
  name         = each.value.subject
  relation_vz_rs_subj_filt_att =
  [aci_filter.terraform_filter[each.value.filter].id]
}
```

これらのリソース定義でvar.contractsは、さまざまなコントラクト、サブジェクト、およびフィルタのマップを含む変数を再び使用していることがわかります。

7. ファイルcontractsから変数を表示しvariable.tfます。

```
variable "contracts" {
  description = "Create contracts with these filters"
  type        = map
  default     = {
    contract_web = {
      contract = "web",
      subject  = "https",
      filter   = "filter_https"
    },
    contract_sql = {
      contract = "sql",
      subject  = "sql",
      filter   = "filter_sql"
    }
  }
}
```

## ステップ 4: プランを適用し、コントラクト/フィルター/ポリシーリソースを作成します

Terraformプランのコントラクト/フィルターリソースに精通したので、次はこのプランを適用します。

前の演習では、構成を適用する前に確認が必要なterraformapplyを使用しました。

-auto-approve CLI引数を使用すると、この確認をバイパスできます。 CLIコマンドを使用すると、変数の値を-var引数に渡すことにより、変数を直接オーバーライドすることもできます。 この場合、それを使って一意のテナント名を渡します。

```
$ terraform apply -auto-approve -var 'tenant=jt_1234'
aci_tenant.terraform_tenant: Creating...
aci_tenant.terraform_tenant: Creation complete after 2s [id=uni/tn-jt_1234]
aci_filter.terraform_filter["filter_sql"]: Creating...
aci_filter.terraform_filter["filter_https"]: Creating...
aci_vrf.terraform_vrf: Creating...
aci_contract.terraform_contract["contract_web"]: Creating...
aci_contract.terraform_contract["contract_sql"]: Creating...
aci_contract.terraform_contract["contract_sql"]: Creation complete after 2s
[id=uni/tn-jt_1234/brc-sql]
aci_contract.terraform_contract["contract_web"]: Creation complete after 2s
[id=uni/tn-jt_1234/brc-web]
```

```

aci_filter.terraform_filter["filter_https"]: Creation complete after 2s
[id=uni/tn-jt_1234/flt-https]
aci_filter.terraform_filter["filter_sql"]: Creation complete after 2s
[id=uni/tn-jt_1234/flt-sql]
aci_filter_entry.terraform_filter_entry["filter_sql"]: Creating...
aci_filter_entry.terraform_filter_entry["filter_https"]: Creating...
aci_contract_subject.terraform_contract_subject["contract_sql"]: Creating...
aci_contract_subject.terraform_contract_subject["contract_web"]: Creating...
aci_filter_entry.terraform_filter_entry["filter_sql"]: Creation complete after 1s
[id=uni/tn-jt_1234/flt-sql/e-sql]
aci_filter_entry.terraform_filter_entry["filter_https"]: Creation complete after 1s
[id=uni/tn-jt_1234/flt-https/e-https]
aci_vrf.terraform_vrf: Creation complete after 5s [id=uni/tn-jt_1234/ctx-prod_vrf]
aci_bridge_domain.terraform_bd: Creating...
aci_contract_subject.terraform_contract_subject["contract_sql"]: Creation complete
after 3s [id=uni/tn-jt_1234/brc-sql/subj-sql]
aci_contract_subject.terraform_contract_subject["contract_web"]: Creation complete
after 3s [id=uni/tn-jt_1234/brc-web/subj-https]
aci_bridge_domain.terraform_bd: Creation complete after 4s
[id=uni/tn-jt_1234/BD-prod_bd]
aci_subnet.terraform_bd_subnet: Creating...
aci_subnet.terraform_bd_subnet: Creation complete after 1s
[id=uni/tn-jt_1234/BD-prod_bd/subnet-[10.10.101.1/24]]

```

Apply complete! Resources: 12 added, 0 changed, 0 destroyed.  
\$

この場合、terraform applyには計画出力が表示されていないことがわかり、必要なインフラストラクチャコンポーネントの作成に直接進むことができます。

APIC GUIでテナントに戻り、次の手順に従って、テナントが存在することを確認します。

- 「contracts」 フォルダを展開します
- 「Standard」 フォルダを展開して、作成したコントラクトを表示します。

The screenshot shows the Cisco Application Policy Infrastructure Controller (APIC) GUI. The top navigation bar includes tabs for System, Tenants, Fabric, Virtual Networking, L4-L7 Services, Admin, Operations, Apps, and Integrations. The 'Tenants' tab is selected, showing the 'ALL TENANTS' view. A search bar at the top right contains the text 'name or descr' with 'js\_10018' entered. Below the search bar, there are filters for 'common', 'js\_10018', 'mgmt', and 'infra'. The main left sidebar shows the tenant structure under 'js\_10018': Quick Start, js\_10018 (selected), Application Profiles, Networking, Contracts (which is expanded and highlighted with a red box), Filters, Policies, and Services. The 'Contracts' section on the right lists two contracts: 'sql' and 'web'. The 'sql' contract is associated with 'VRF' scope, 'Unspecified' QoS Class, 'Unspecified' Target DSCP, and subjects 'sql'. The 'web' contract is also associated with 'VRF' scope, 'Unspecified' QoS Class, 'Unspecified' Target DSCP, and subjects 'https'.

図1-60. ネットワーク構成を検証する

## ステップ 5: ACI Terraform アプリケーションプロファイルリソースを確認する

EPGを使用してアプリケーションプロファイルを作成するには、次の四つのリソースが必要です。

- `aci_application_profile`
  - `aci_application_epg`
  - `aci_epg_to_domain`
  - `aci_epg_to_contract`
1. コードサンプルディレクトリ`aci_terraform_learning_labs_code_samples/application_policies`にディレ

クトリを変更します。

2. ディレクトリを初期化するためにterraform initを実行します。
3. main.tfファイルを開くと、各リソースについて説明し、計画で提供されている例を確認します。

## アプリプロファイルリソース

アプリケーションプロファイルの管理は、aci\_application\_profileリソースを使用して行われます。ACIでは、アプリケーションプロファイルは、アプリケーションを構成するサービスのコンテナとして使用されます。それらはより組織的であるnameため、2つの引数のみがあります。アプリケーションプロファイルが存在することを確認するために使用され、tenant\_dnそれが存在するテナントを指定するために使用されます。

4. aci\_application\_profile計画のリソースを確認します。

プランの最初の新しいリソースは、EPGのアプリプロファイルが存在することを確認するために使用されます。

```
# Define an ACI Application Profile Resource.
resource "aci_application_profile" "terraform_ap" {
  tenant_dn = aci_tenant.terraform_tenant.id
  name      = var.ap
  description = "App Profile Created Using Terraform"
}
```

## EPGリソース

aci\_application\_epgリソースは、アプリケーションのエンドポイントグループを管理するために使用されます。EPGにはいくつかの議論があり、リソースがサポートするいくつかを強調します。

- application\_profile\_dn: EPGが属するアプリケーションプロファイルのDN。
  - name: EPGの名前。
  - relation\_fv\_rs\_bd: EPGに属するホストが関連付けられているブリッジドメインのDN。これにより、ホストが接続されているブロードキャストドメインと、ホストが使用できるIPサブネットが決まります。
5. aci\_application\_epg計画のリソースを確認します

プランの2番目の新しいリソースは、アプリケーションに「web」および「db」EPGが存在することを確認するために使用されます。

```
resource "aci_application_epg" "terraform_epg" {
  for_each           = var.epgs
  application_profile_dn = aci_application_profile.terraform_ap.id
  name              = each.value.epg
  relation_fv_rs_bd     = aci_bridge_domain.terraform_bd.id
  description        = "EPG Created Using Terraform"
}
```

このリソースを見ると、for\_each行がepgsという名前の変数を使用していることがわかります。

- まず、variable.tfを開くと、プランの変数を含むファイルが表示されます。この演習では、epgsとepg\_contractsを追加しました。
- 次に、値をこのラボで使用しているテナントの名前tenantに変更します。

```
variable "epgs" {
  description = "Create epg"
  type        = map
  default     = {
    web_epg = {
      epg   = "web",
      bd    = "prod_bd",
      encap = "21"
    },
    db_epg = {
      epg   = "db",
      bd    = "prod_bd",
      encap = "22"
    }
  }
}

variable "epg_contracts" {
  description = "epg contracts"
  type        = map
  default     = {
```

```

    terraform_one = {
      epg        = "web_epg",
      contract   = "contract_web",
      contract_type = "provider"
    },
    terraform_two = {
      epg        = "web_epg",
      contract   = "contract_sql",
      contract_type = "consumer"
    },
    terraform_three = {
      epg        = "db_epg",
      contract   = "contract_sql",
      contract_type = "provider"
    }
  }
}

```

## EPGとドメインバインディング

このaci\_epg\_to\_domainリソースは、EPGからドメインへのバインディング(VMMおよび物理)を管理するために使用されます。リソースのコンポーネントは次のとおりです。

- application\_epg\_dn ドメインに関連付けるEPGのDN
- tdn EPGに関連付けるVMMまたは物理ドメインのDN

```

# Associate the EPG Resources with a VMM Domain.
resource "aci_epg_to_domain" "terraform_epg_domain" {
  for_each      = var.epgs
  application_epg_dn = aci_application_epg.terraform_epg[each.key].id
  provider_profile_dn = "uni/vmmp-VMware/dom-aci_terraform_lab"
}

```

このリソースもepgs前のリソースと同じ変数を使用していることに注意してください。これにより、構成データを2回書き直す必要がなくなり、リソース間で値の一貫性を保つことができます。

## EPGをコントラクトにバインドする

このaci\_epg\_to\_contractリソースは、EPGに関連付けられているコントラクトを管理するために使用されます。このリソースは次の引数を使用します。

- application\_epg\_dn コントラクトに関連付けるEPGのDN
- contract\_dn EPGに関連付けられるコントラクトのDN
- contract\_type この引数は、EPGがコントラクトであるproviderかconsumerコントラクトであるかを示すために使用
- 6. aci\_epg\_to\_contractプラン内のリソースとepg\_contracts、for\_eachメタ引数によって使用される変数を確認します。

```

# Associate the EPGs with the contracts
resource "aci_epg_to_contract" "terraform_epg_contract" {
  for_each      = var.epg_contracts
  application_epg_dn = aci_application_epg.terraform_epg[each.value.epg].id
  contract_dn   = aci_contract.terraform_contract[each.value.contract].id
  contract_type = each.value.contract_type
}

variable "epg_contracts" {
  description = "epg contracts"
  type        = map
  default     = {
    terraform_one = {
      epg        = "web_epg",
      contract   = "contract_web",
      contract_type = "provider"
    },
    terraform_two = {
      epg        = "web_epg",
      contract   = "contract_sql",
      contract_type = "consumer"
    },
    terraform_three = {
      epg        = "db_epg",
      contract   = "contract_sql",
      contract_type = "provider"
    }
  }
}

```

```

        contract_type = "provider"
    }
}
}

```

## ステップ 6: プランを適用し、アプリケーションポリシーを作成します

Terraformプランのアプリケーションリソースに精通したので、次はこのプランを適用します。

前の演習と同様に、-var引数を使用してテナント名を渡します。

```

$ terraform apply -auto-approve -var 'tenant=jt_1234'
aci_tenant.terraform_tenant: Creating...
aci_tenant.terraform_tenant: Creation complete after 2s [id=uni/tn-jt_1234]
aci_filter.terraform_filter["filter_https"]: Creating...
aci_application_profile.terraform_ap: Creating...
aci_vrf.terraform_vrf: Creating...
aci_filter.terraform_filter["filter_sql"]: Creating...
aci_contract.terraform_contract["contract_sql"]: Creating...
aci_contract.terraform_contract["contract_web"]: Creating...
aci_application_profile.terraform_ap: Creation complete after 1s
[id=uni/tn-jt_1234/ap-intranet]
aci_contract.terraform_contract["contract_sql"]: Creation complete after 1s
[id=uni/tn-jt_1234/brc-sql]
aci_contract.terraform_contract["contract_web"]: Creation complete after 1s
[id=uni/tn-jt_1234/brc-web]
aci_filter.terraform_filter["filter_sql"]: Creation complete after 1s
[id=uni/tn-jt_1234/flt-sql]
aci_filter.terraform_filter["filter_https"]: Creation complete after 1s
[id=uni/tn-jt_1234/flt-https]
aci_filter_entry.terraform_filter_entry["filter_sql"]: Creating...
aci_contract_subject.terraform_contract_subject["contract_web"]: Creating...
aci_contract_subject.terraform_contract_subject["contract_sql"]: Creating...
aci_filter_entry.terraform_filter_entry["filter_https"]: Creating...
aci_filter_entry.terraform_filter_entry["filter_sql"]: Creation complete after 1s
[id=uni/tn-jt_1234/flt-sql/e-sql]
aci_filter_entry.terraform_filter_entry["filter_https"]: Creation complete after
1s [id=uni/tn-jt_1234/flt-https/e-https]
aci_contract_subject.terraform_contract_subject["contract_web"]: Creation
complete after 2s [id=uni/tn-jt_1234/brc-web/subj-https]
aci_contract_subject.terraform_contract_subject["contract_sql"]: Creation
complete after 2s [id=uni/tn-jt_1234/brc-sql/subj-sql]
aci_vrf.terraform_vrf: Creation complete after 3s
[id=uni/tn-jt_1234/ctx-prod_vrf]
aci_bridge_domain.terraform_bd: Creating...
aci_bridge_domain.terraform_bd: Creation complete after 5s
[id=uni/tn-jt_1234/BD-prod_bd]
aci_subnet.terraform_bd_subnet: Creating...
aci_application_epg.terraform_epg["db_epg"]: Creating...
aci_application_epg.terraform_epg["web_epg"]: Creating...
aci_subnet.terraform_bd_subnet: Creation complete after 3s
[id=uni/tn-jt_1234/BD-prod_bd/subnet-[10.10.101.1/24]]
aci_application_epg.terraform_epg["web_epg"]: Creation complete after 6s
[id=uni/tn-jt_1234/ap-intranet/epg-web]
aci_application_epg.terraform_epg["db_epg"]: Creation complete after 6s
[id=uni/tn-jt_1234/ap-intranet/epg-db]
aci_epg_to_contract.terraform_epg_contract["terraform_one"]: Creating...
aci_epg_to_contract.terraform_epg_contract["terraform_two"]: Creating...
aci_epg_to_domain.terraform_epg_domain["db_epg"]: Creating...
aci_epg_to_contract.terraform_epg_contract["terraform_three"]: Creating...
aci_epg_to_domain.terraform_epg_domain["web_epg"]: Creating...
aci_epg_to_contract.terraform_epg_contract["terraform_one"]: Creation complete
after 4s [id=uni/tn-jt_1234/ap-intranet/epg-web/rsprov-web]
aci_epg_to_contract.terraform_epg_contract["terraform_three"]: Creation
complete after 4s [id=uni/tn-jt_1234/ap-intranet/epg-db/rsprov-sql]
aci_epg_to_contract.terraform_epg_contract["terraform_two"]: Creation complete
after 4s [id=uni/tn-jt_1234/ap-intranet/epg-web/rscons-sql]
aci_epg_to_domain.terraform_epg_domain["web_epg"]: Creation complete after 4s
[id=uni/tn-jt_1234/ap-intranet/epg-web/rsdomAtt-[uni/vmmp-VMware/dom-aci_terrafor
m_lab]]

```

```
aci_epg_to_domain.terraform_epg_domain["db_epg"] : Creation complete after 4s  
[id=uni/tn-jt_1234/ap-intranet/epg-db/rsdomAtt-[uni/vmmp-VMware/dom-aci_terraform_lab]]
```

Apply complete! Resources: 20 added, 0 changed, 0 destroyed.

\$

terraform applyにより、必要なインフラストラクチャコンポーネントが作成されたことがわかります。

APIC GUIでテナントに戻り、次の手順に従って、テナントが存在することを確認します。

- 「Application Profile [アプリケーションプロファイル]」 フォルダを開いています
- 「Application EPG [アプリケーションEPG]」 フォルダを開いて、作成したEPGを表示します。

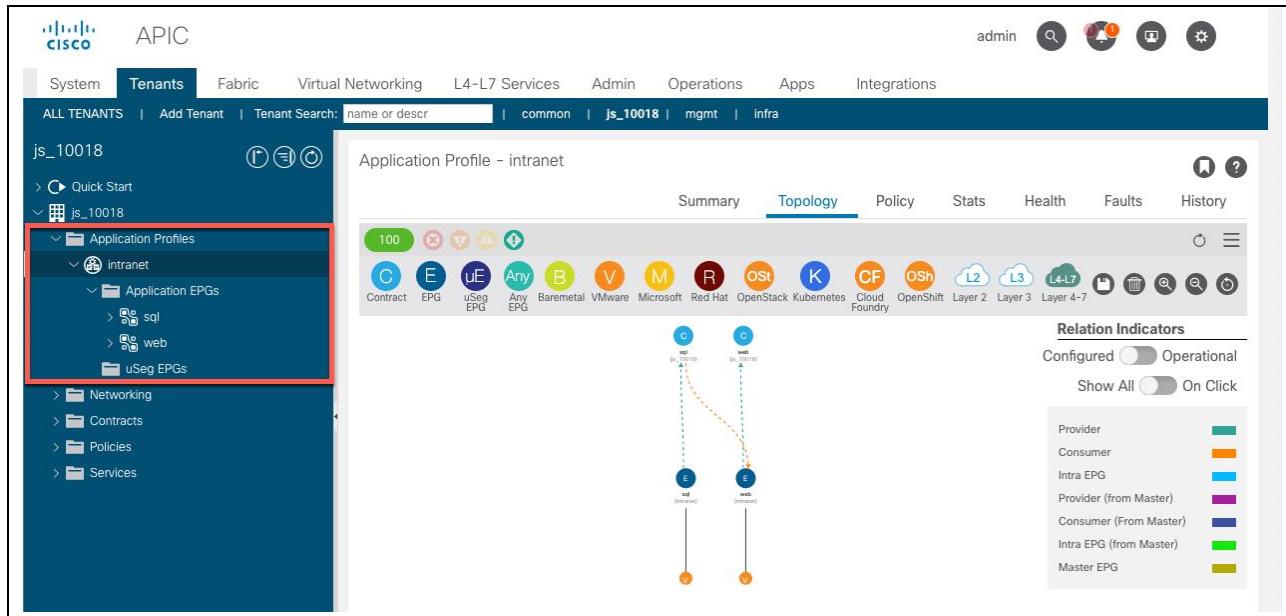


図1-61. ネットワーク構成を検証する

お疲れ様でした！

すばらしい仕事です。次のラボでは、リソースが利用できない場合のケースの処理方法を見ていきます。

## 5.3. ACI REST API を汎用 ACI Terraform リソースをラップする

### 概要

Cisco Application Centric Infrastructure (ACI)には、APIを使用してプログラムでACIファブリックを構築および運用するのに役立つツールがいくつかあります。これらのツールには、APIインスペクター、ACI Cobra SDK、ACI Toolkit、Ansibleなどがあります。これらのツールはすべて、他のDevNet LearningLabsでカバーされています。

この学習モジュールのラボでは、Terraformを使用してACIを自動化する最新の方法を使用して説明します。Terraform、特にACI用のTerraformリソースとデータソースを使用する利点は、カスタムスクリプトを作成および開発する必要がなくなることです。Terraformリソースは、基本的に、特定のタスクを実行する事前に作成されたコードのブロックです。これにより、APIリクエストの調査、適切なAPIリクエストを行うためのコードの開発、コードのテストとデバッグにかかる時間を節約できます。

### 目的

- 一般的なACIリソースを使用する

### ACI Generic Resourceを確認

ACI Terraformプロバイダは、現在のモジュールが特定の機能をサポートしていないギャップを埋めるために、汎用のACIリソースaci\_restを提供します。

このリソースのオプションについて説明してから、Terraformを使用して、専用のリソースを持たないACIオブジェクトをプロビジョニングします。

注：続行する前に、コードサンプルディレクトリaci\_terraform\_learning\_labs\_code\_samples/generic\_restにあり、PATHにterraformバイナリがあることを確認してください。

### ステップ 1: 一般的な ACI Terraform リソースを確認する

Terraform ACIプロバイダは、ACIで最も一般的に管理されるオブジェクトをサポートするためのリソースを提供します。aci\_restリソースは、現在のリソースが特定の機能をサポートしていないギャップを埋めるために作成されました。このリソースを使用してAPICにAPIリクエストを行う方法を紹介します。このAPIに慣れていない場合は、DevNetから入手できるACIラボを確認してください。

### RESTリソース

aci\_restリソースは、YAMLまたはJSONのいずれかを使用してAPICにPOST(適用された場合)またはDELETE(破棄された場合)要求を行う方法を提供します。このリソースには、注意すべき3つの重要なパラメータがあります。

- path:-オブジェクトを作成するパス。ACI APIのオブジェクトのDNに相当します。
- content -オブジェクトの属性として使用されるキーと値のペアのマップ。
- payload-パスに追加されたRESTエンドポイントに直接渡すことができるフリースタイルのJSONまたはYAMLペイロード。コンテンツまたはペイロードのいずれかが必要です。

構成オプションの完全なリストは、ドキュメントに記載されています。

aci\_terraform\_learning\_labs\_code\_samplesリポジトリディレクトリのgeneric\_restフォルダにあるmain.tfファイルを開きます。2番目のリソースは、前のラボで使用したテナントのl3outを表す汎用aci\_restリソースです。#テナントにL3外部ネットワークがあることを確認してコメントされたリソースを見てください。

```
# Ensure Tenant has L3 external network.
resource "aci_rest" "terraform_l3out" {
    path    = "api/mo/${aci_tenant.terraform_tenant.id}/out-${var.l3out.name}.json"
    payload = <<EOF
    {"l3extOut": {"attributes": {"descr":${var.l3out.description},
    "name":${var.l3out.name}}}}
    EOF
}
```

このリソース定義では、変数var.l3out.name、var.l3out.descriptionを使用します。これは、Terraform変数オブジェクトを使用して、より複雑な変数をプランに提供する方法を示します。このアプローチの利点は、別 のL3Outを使用するたびにファイルを手動で編集しなくとも、planを再利用できることです。

また、補間を使用して、テナントリソースインスタンス変数を使用してテナントDNを参照します。

Terraformを使用すると、planで使用する変数をvariable.tfという別のファイルで定義できます。

variable.tfファイルで、l3outオブジェクト変数を調べます。

```

variable "tenant" {
  type    = string
  default = "terraform-tenant"
}
variable "l3out" {
  type    = object({
    name    = string
    description = string
  })
  default = {
    name    = "corp_l3"
    description = "Created Using Terraform"
  }
}

```

テナントを一意にするには、tenant変数をイニシャルのような一意の文字列に変更し、その後に次のような電話番号を続けます。

```

variable "tenant" {
  type    = string
  default = "jt_1234"
}
variable "l3out" {
  type    = object({
    name    = string
    description = string
  })
  default = {
    name    = "corp_l3"
    description = "Created Using Terraform"
  }
}

```

## ステップ 2: 計画を適用し、一般的な ACI Terraform リソースを実行します

Terraformプランの一般的なACI Terraformリソースに精通したので、次はこのプランを適用します。

このモジュールの2番目のラボでは、テナント、VRF、ブリッジドメイン、サブネット、コントラクト、およびフィルターを作成しました。このplanを実行すると、テナントに新しいL3Outが作成されます。

前のセクションで説明したmain.tfを含む

ディレクトリaci\_terraform\_learning\_labs\_code\_samples/generic\_resourceにいることを確認してください。

正しいディレクトリにいるときに、次のコマンドを実行します。

```
$ terraform init
```

「Terraform has been successfully initialized! [Terraformが正常に初期化されました!]」と表示されます。このコマンドを実行した後、他の情報とともにメッセージを表示します。

次に、次のapplyコマンドを実行できます。

```
$ terraform apply
```

```
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:

```

# aci_rest.terraform_l3out will be created
+ resource "aci_rest" "terraform_l3out" {
  + class_name = (known after apply)
  + dn        = (known after apply)
  + id        = (known after apply)
  + path      = (known after apply)
  + payload   = <<~EOT
    {"l3extOut": {"attributes": {"descr": "Created Using Terraform", "name": "corp_l3"} }}
    EOT
}

# aci_tenant.terraform_tenant will be created

```

```
+ resource "aci_tenant" "terraform_tenant" {
    + annotation = "orchestrator:terraform"
    + description = "This tenant is created by terraform"
    + id         = (known after apply)
    + name       = "jt_1234"
    + name_alias = (known after apply)
}
```

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

**Do** you want **to** perform these actions?  
Terraform will perform the actions described above.  
**Only** 'yes' will be accepted to approve.

Enter a value:

terraform applyは、terraform planの出力を表示し、**yes**と入力して**Enter**キーを押して確認するように求めます。

Enter a value: **yes**

```
aci_tenant.terraform_tenant: Creating...
aci_tenant.terraform_tenant: Creation complete after 2s [id=uni/tn-jt_1234]
aci_rest.terraform_l3out: Creating...
aci_rest.terraform_l3out: Creation complete after 0s [id={}]
```

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

\$

APIC GUIでテナントに戻り、次の手順に従ってL3Outが存在することを確認します。

- 「Networking」 フォルダを展開します
- 「L3Outs」 (または「ExternalRouted Networks」) フォルダを展開して、作成したcorp\_l3l3Outを表示します。

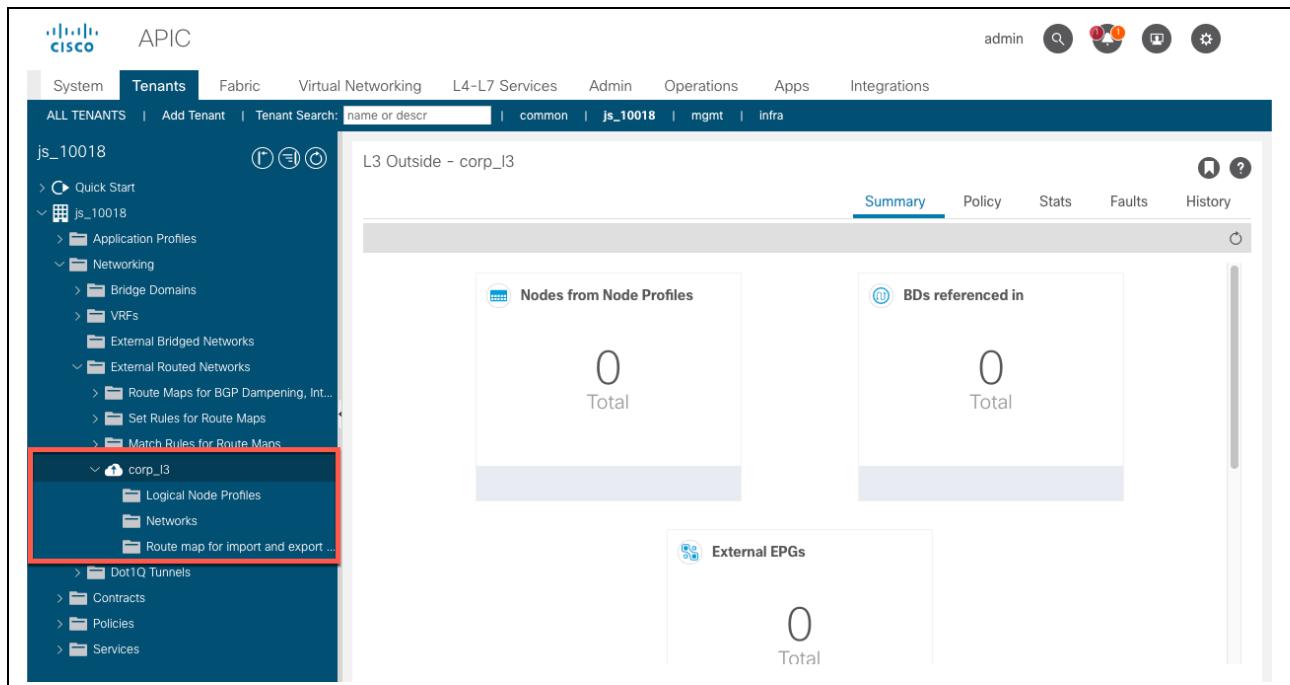


図1-62. L3Out構成を検証する

### ステップ 3: 計画を破棄してラボをクリーンアップします

このラボの構成をクリーンアップするには、テナントにDELETEリクエストを発行しplanを破棄します。

前のセクションで説明したmain.tfを含む

ディレクトリaci\_terraform\_learning\_labs\_code\_samples/generic\_resourceにいることを確認してください。

正しいディレクトリにいるときに、次のコマンドを実行します。

\$ **terraform init**

「Terraform has been successfully initialized! [Terraformが正常に初期化されました!]」と表示されます。このコマンドを実行した後、他の情報とともにメッセージを表示します。

次に、destroyコマンドを実行できます。

```
$ terraform destroy
aci_tenant.terraform_tenant: Refreshing state... [id=uni/tn-terraform-tenant]
aci_rest.terraform_l3out: Refreshing state... [id={}]
```

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

Terraform will perform the following actions:

```
# aci_rest.terraform_l3out will be destroyed
- resource "aci_rest" "terraform_l3out" {
    - id      = jsonencode({}) -> null
    - path    = "api/mo/uni/tn-terraform-tenant/out-corp_l3.json" -> null
    - payload = <<~EOT
    {"l3extOut": {"attributes": {"descr":Created Using Terraform, "name":corp_l3}}}
    EOT -> null
}

# aci_tenant.terraform_tenant will be destroyed
- resource "aci_tenant" "terraform_tenant" {
    - annotation        = "orchestrator:terraform" -> null
    - description       = "This tenant is created by terraform" -> null
    - id                = "uni/tn-terraform-tenant" -> null
    - name              = "terraform-tenant" -> null
    - relation_fv_rs_tn_deny_rule = [] -> null
}
```

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

Do you really want to destroy all resources?

Terraform will destroy all your managed infrastructure, as shown above.  
There is no undo. Only 'yes' will be accepted to confirm.

Enter a value:

terraform applyは、terraform planの出力を表示し、**yes**と入力して**Enter**キーを押して確認するように求めます。

Enter a value: **yes**

```
aci_rest.terraform_l3out: Destroying... [id={}]
aci_rest.terraform_l3out: Destruction complete after 1s
aci_tenant.terraform_tenant: Destroying... [id=uni/tn-terraform-tenant]
aci_tenant.terraform_tenant: Destruction complete after 0s
```

Destroy complete! Resources: 2 destroyed.

\$

**お疲れ様でした！**

素晴らしい仕事です！これで、ACI Terraformリソースを使用して、ACI環境全体を「as Code [コードとして]」管理する方法をしっかりと理解できました。