Ansible によるシスコ データセンターの自動化（v1）

最終更新日：2018 年 10 月 4 日

重要：このドキュメントには、リリース前のソフトウェアに関する内容が含まれています。また、一部の機能で問題が発生する可能性があります。付属のドキュメントは、dCloud が作成したものではなく、dCloud による検証も受けていません。Cisco dCloud で、新しいリリースを定期的にチェックしてください。

このラボについて

このドキュメントは、事前設定済みの Ansible® によるシスコ データセンターの自動化（v1）のガイドであり、以下の内容を含んでいます。

要件

次の表に、このデモンストレーションの要件の概要を示します。

表 1. 要件

必須	オプション
● Cisco dCloud 用に登録および設定が済んでいること ● ラップトップ	● Cisco AnyConnect®

コンポーネント

Cisco VIRL

NXOSv 7.0 (3) I7(1)

ACI シミュレータ 2.2(2e)

Ansible ホスト - CentOS 7

PC ワークステーション（198.18.133.252）

- Ansible ホスト、NXOSv1、NXOSv2 への PuTTY ショートカットを含む
- Chrome で、コントロール センター起動、ライブ ビジュアル化エンジン、ACI シミュレータのタブが自動で開きます。

機能

Cisco VIRL	ネットワーク トポロジを構築するための仮想環境 ネットワー キング コンポーネントのシミュレーション シスコ オペレーティング システム（IOS XE、IOS Classic、IOS-XR、NX-OS）が稼働している複数の仮想マシン（VM）を実行可能 サードパーティ VM を実行可能 任意のノード上でネットワーク トラフィックを取得および分析 物理機器の導入に先立って設定の検証が可能
NXOS	モジューラ化された OS であり、さまざまな要素にアクセスすることが可能

	Puppet、Chef、Ansible などの組み込み DevOps 自動化ツールや業界標準 YANG モデルをネイティブにサポート サードパーティ製のアプリケーションを NXOS アプリケーション ホスティングが可能 NXOS が提供する API によってオーケストレーション ツールとの統合が可能
ACI	業界をリードする Software Defined Networking（SDN）ソリューションの Cisco ACI によって、アプリケーションの迅速な導入とデータセンターの自動化を実現 ファブリックおよびそのポリシーのプロビジョニングの自動化に使用できる、さまざまなプログラミングのオプションを用意 ACI 自動化のためのオプションとして、ネイティブの REST API、ACI Python SDK、ACI Toolkit、ACI Ansible モジュールなどを利用可能

ラボ演習について

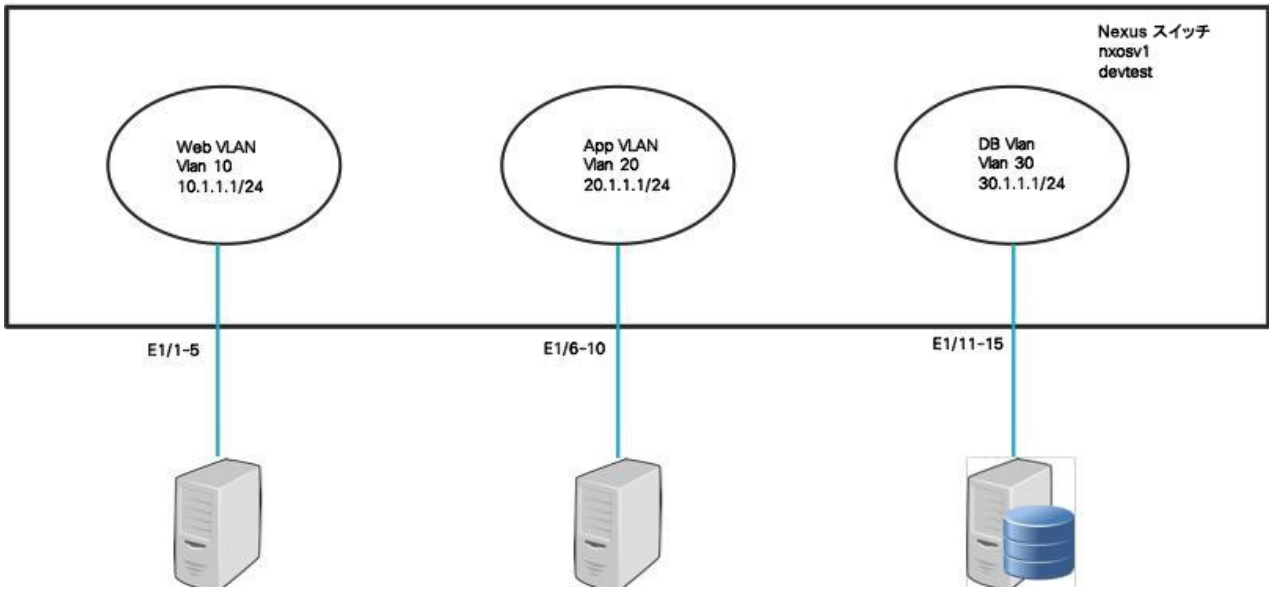
このドキュメントでは、オープン ソースの Ansible 自動化テクノロジー（http://ansible.com）を活用して Cisco NX-OS と Cisco ACI インフラストラクチャを自動化する方法を、手順ごとに説明します。ラボ タスクの実行や実験のための環境として、Linux、NXOS、ACI の仮想化インスタンスを使用します。

Ansible を利用すると、コンピューティング、ストレージ、ネットワークなどインフラストラクチャ内のさまざまな部分にまたがるタスクを、簡単に自動化することができます。Ansible の一般的なサーバ運用では、管理対象デバイス上で Python コードがローカルで実行されますが、Cisco NX-OS デバイスで Ansible を実行する場合、NXAPI が使用されます。運用がこのように一元化されたモデルは、大規模ネットワークのオペレータにとってメリットが大きく、サービス プロバイダー業界において Ansible が人気を得ている理由の 1 つになっています。

今回の演習では、2 つの仮想 NXOS デバイス（nxosv1 および nxosv2）を設定します。このシナリオでは、nxosv1 を *devtest* デバイス、nxosv2 を *production* デバイスとして使用します。これらを利用して、Ansible が持つ数多くの高度な機能を説明していきます。NXOS のラボ演習では、さまざまなプライベート VLAN とパブリック VLAN、IP インターフェイス、IP ルート テーブル、VRF を設定および操作します。ACI では、テナント、ブリッジ ドメイン（BD）、VRF、コントラクト、エンドポイント グループ（EPG）、アプリケーション プロファイルを設定します。

まず、基本的な VLAN と IP インターフェイスを使用した *devtest* デバイスの簡単な設定から始めます。それを通して、Ansible の基本的な機能と構造や、ほとんどのネットワーク専門家になじみのある一般的な使用例を紹介します。

図 1. devtest デバイス用のオブジェクト構成



後半の NXOS ラボ演習では、一般的な構成を土台にして特定のネットワークインフラストラクチャの役割に合わせたカスタマイズや拡張を行う方法を取り上げます。たとえば、production デバイスでは、以下を追加で実施して、セキュリティを向上させることができます。

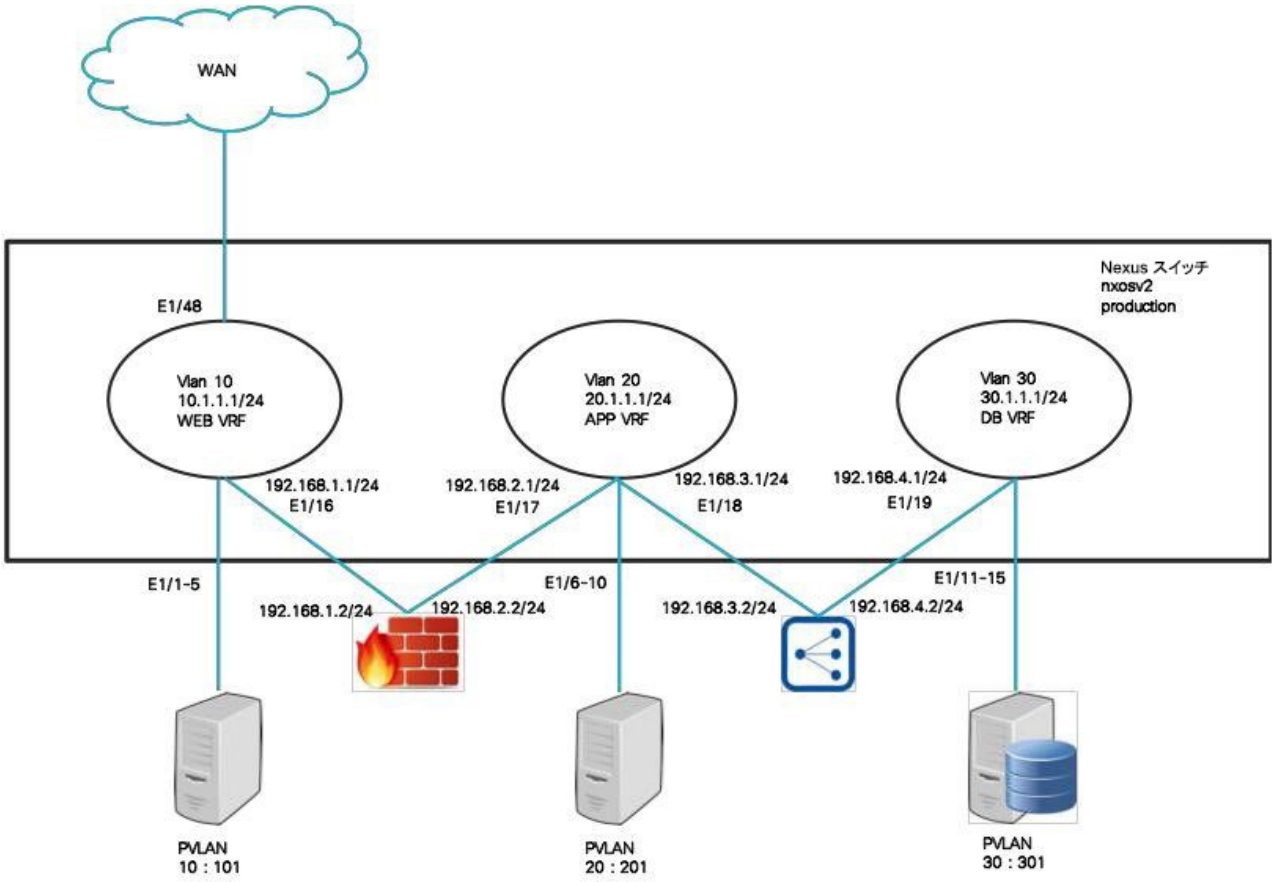
VRF（Web、アプリ、DB）を作成し、FW やロード バランサを使用して連携させる。

Web VLAN の IP アクセス リストを作成する。

トラフィックが確実にファイアウォールやロード バランサを通過できるように、必要なスタティック ルートを設定する。

各 VRF 内でセキュリティを確保するために、プライベート VLAN を使用する。

図 2. production デバイス用のオブジェクト構成



注：最初の 3 つのラボ演習では、devtest デバイスである nxosv1 のみを設定します。ラボ 4 では、Ansible の高度な用法を使用して、devtest デバイス（nxosv1）と production（nxosv2）デバイスの両方を設定します。

トポロジ

このコンテンツには、スクリプト形式のシナリオと、ソリューションの機能を実例で示すために事前設定された管理ユーザとコンポーネントが含まれています。コンポーネントのほとんどは、管理ユーザ アカウントを使用して任意の設定が可能です。コンポーネントへのアクセスに使用する IP アドレスとユーザ アカウント情報は、アクティブ セッションの [トポロジ（Topology）] メニューのコンポーネント アイコンをクリックして確認するか、それらを必要とするシナリオ内の手順で確認できます。

図 3. dCloud のトポロジ

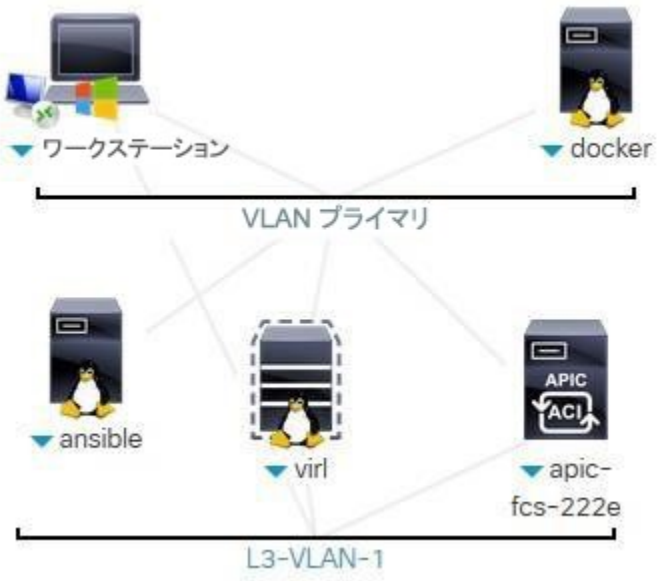


表 2. 機器の詳細

名前	説明	ホスト名（FQDN）	IP アドレス	ユーザ名	パスワード
ワークステーション	ローカル ネットワーク アクセスが可能な Windows VM		198.18.133.252	Administrator	C1sco12345
Ansible	Centos VM（Ansible playbook 実行用）	ansible	198.18.134.50	ansible	C1sco12345
apic-fcs-222e	ACI シミュレータ	apic	198.18.133.200	admin	C1sco12345
VIRL	NXOS シミュレーション デバイス	nxosv1、nxosv2	198.18.1.51、52	ansible	ansible

Windows ワークステーション：このラボで使用するその他のコンポーネントと接続している Windows VM です。ラボ内のコンポーネント/デバイスはすべてプライベート ネットワーク上にあり、この Windows ワークステーション経由でしかアクセスできません。ユーザは RDP を利用してこの Windows ワークステーションにアクセスできます。

VIRL：Cisco Virtual Internet Routing Lab（VIRL）は、ネットワーク仮想化およびオーケストレーション用のプラットフォームです。必要な NXOS ラボのトポロジを VIRL でシミュレーションします。VM Maestro は、シミュレーション トポロジを作成して実行するために使用する、フロント エンドのソフトウェアです。

Ansible VM：ユーザがすべてのラボ演習を実行する Centos VM です。この VM は、VIRL デバイスおよび ACI シミュレータに接続されています。この VM には、Ansible、Python、および必要なすべてのパッケージがインストール済みです。ユーザは、Windows ワークステーション上で Putty を使用し、SSH によって Ansible VM にアクセスできます。

ACI シミュレータ：ACI インフラストラクチャのシミュレーションに使用します。ユーザは、このシミュレータを使用して ACI-Ansible ラボ演習を実施します。

はじめに

プレゼンテーションの前に

Cisco dCloud では、実際の対象者の前でプレゼンテーションを行う前に、アクティブなセッションを使用して、このドキュメントのタスクを実施しておくことを強く推奨します。そうすることで、ドキュメントとコンテンツの構成に慣れることができます。

場合によっては、環境を元の構成にリセットするため、このガイドに従った後に新しいセッションをスケジュールする必要があります。

プレゼンテーションを成功させるには入念な準備が不可欠です。

次の手順に従ってコンテンツのセッションをスケジュールし、プレゼンテーション環境を設定します。

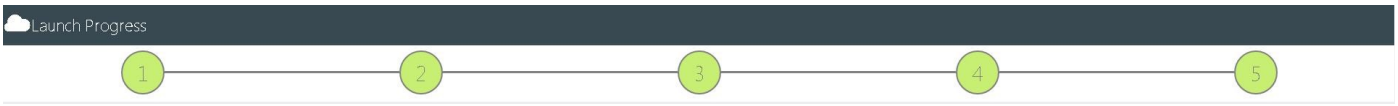
1. dCloud セッションを開始します。[手順を見る] [英語]

注：セッションがアクティブになるまで最長で 20 分かかることがあります。

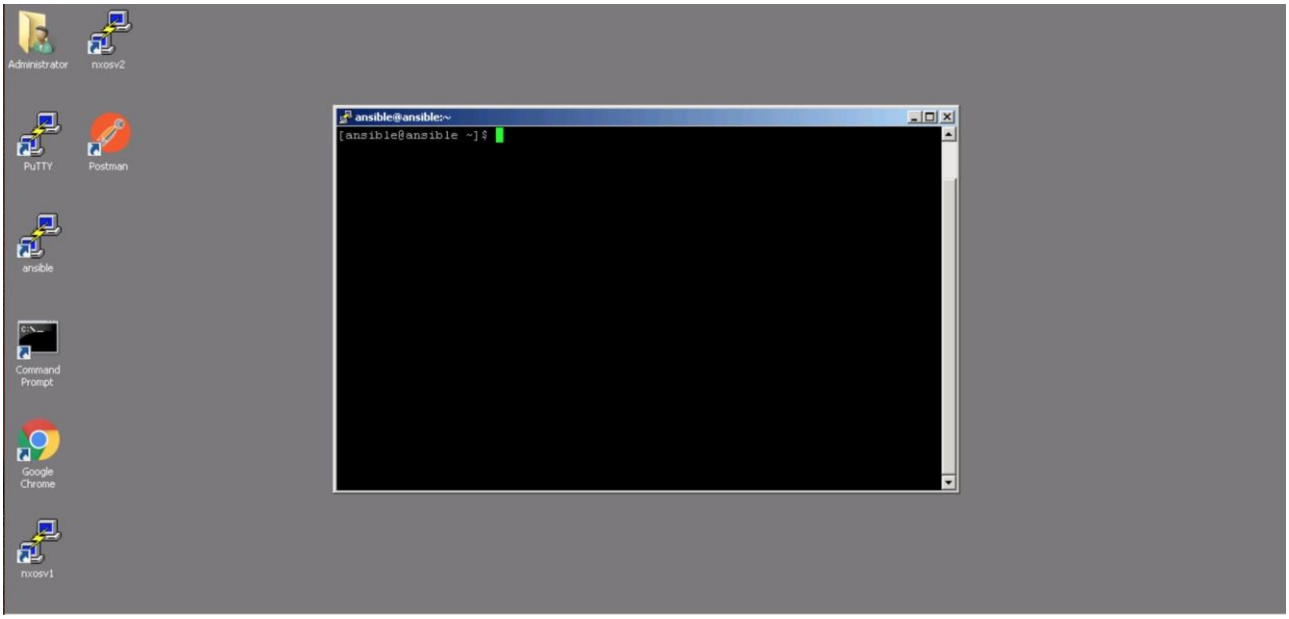
2. 最適なパフォーマンスを得るには、**Cisco AnyConnect VPN** [手順を見る] [英語] および**ラップトップのローカル RDP クライアント** [手順を見る] [英語] を使用してワークステーションに接続します。

   ワークステーション 1：**198.18.133.252**、ユーザ名：**administrator**、パスワード：**C1sco12345**

3. [コントロールセンター起動（Launch Control Center）] 画面ですべての段階が完了していることを確認し、次に進みます。



4. Chrome Web ブラウザを最小化し、**ansible** ターミナル アイコンをダブルクリックします。下図のように Linux シェル セッションが開きます。



5. ラボ演習は Ansible ホスト上で実行します。dc-automation-bootcamp の下の各サブディレクトリに各演習が用意されています。

注：VI エディタの使用方法

ラボ演習では、ansible ホスト内の既存スクリプトの編集が必要になります。編集には VI エディタ ツールを使用します。以下に、VI エディタを使用してスクリプトを編集する手順を示します。ラボ演習でスクリプトの編集が必要になった場合は、このセクションを参照してください。

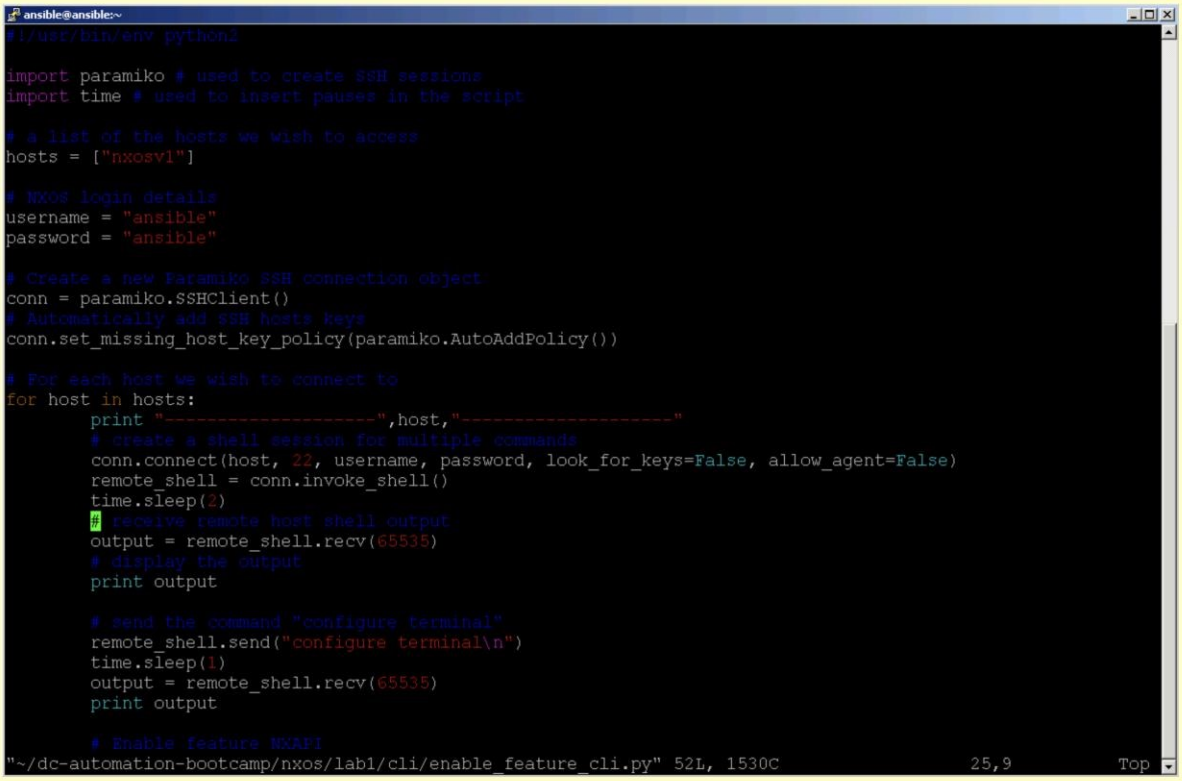
スクリプトを開く：ansible ホストの Putty ターミナルで以下のコマンドを入力し、VI エディタでスクリプトを開きます。

```
[ansible@ansible ~]$ vi <enter path to the script here>
```

例：

```
[ansible@ansible ~]$  vi dc-automation-bootcamp/nxos/lab1/cli/enable_feature_cli.py
```

VI エディタ ウィンドウが開き、スクリプトが表示されます。スクリプトをスクロールするには上矢印/下矢印キーまたは PageUp/PageDown キーを使用します。



スクリプトの編集：編集するスクリプトを開きます。スクリプトが開いたら、キーボードの「i」キーを押します。これにより、編集モードに入ります。編集する行に移動し、変更します。編集モードを終了するには、「Esc」キーを押します。

変更の保存：編集が終わったら、「Esc」キーを押して編集モードを終了します。「:w」と入力し、Enter を押して変更を保存します。

変更を元に戻す：スクリプトに加えた変更を元に戻す場合は、「Esc」キーを押して編集モードを終了します。キーボードの「u」キーを押して、最後に変更した内容を元に戻します。

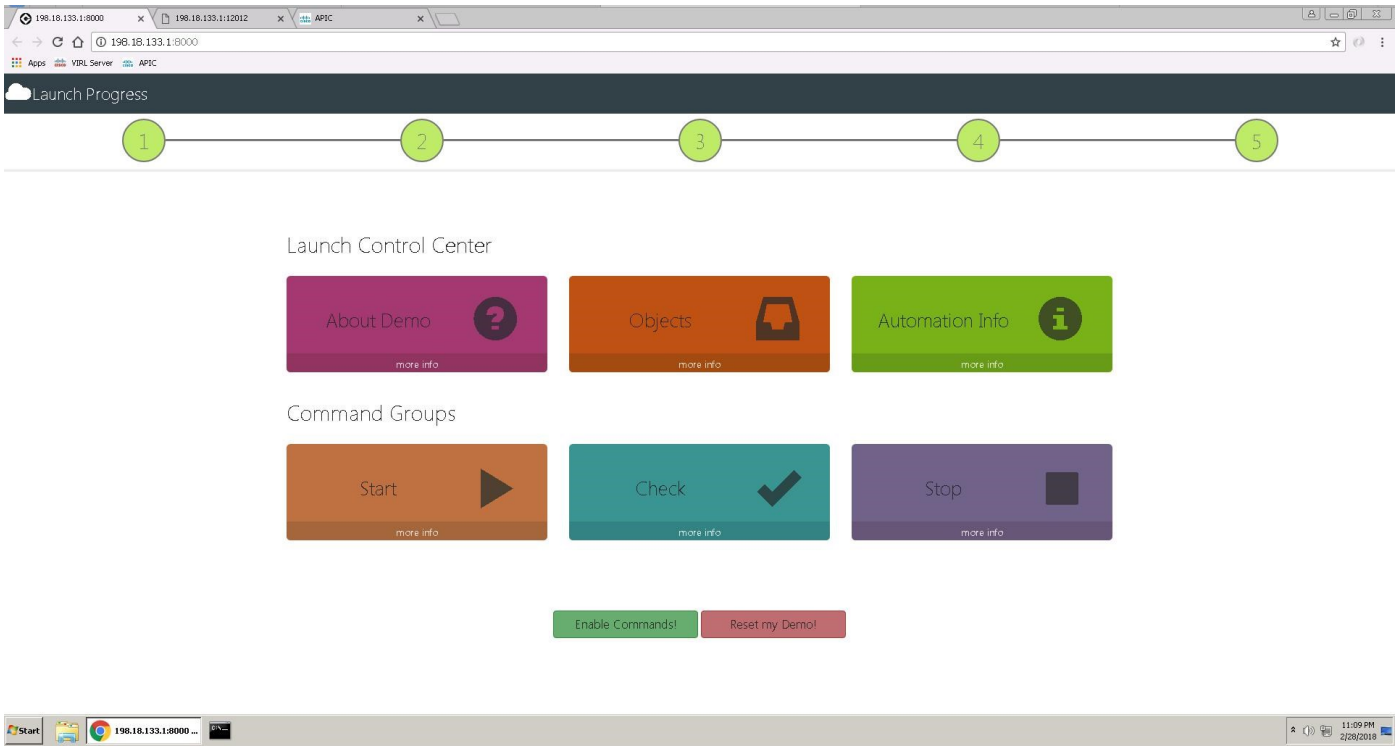
スクリプトの終了：「Esc」キーを押して編集モードを終了したら、「:q」と入力し、Enter を押してスクリプトを終了します。保存していない変更がある場合は、スクリプトを終了する前に変更を保存します。変更を破棄して終了する場合は、「: q!」と入力してから Enter を押して終了します。

トラブルシューティング

VIRL インスタンスに問題が発生した場合は、Windows リモート デスクトップ上の Chrome で利用できる Web ツールを使用してください。

最初のタブでは、VIRL シミュレーションの操作が可能です。シミュレーションの開始、停止、リセットを実施できます。

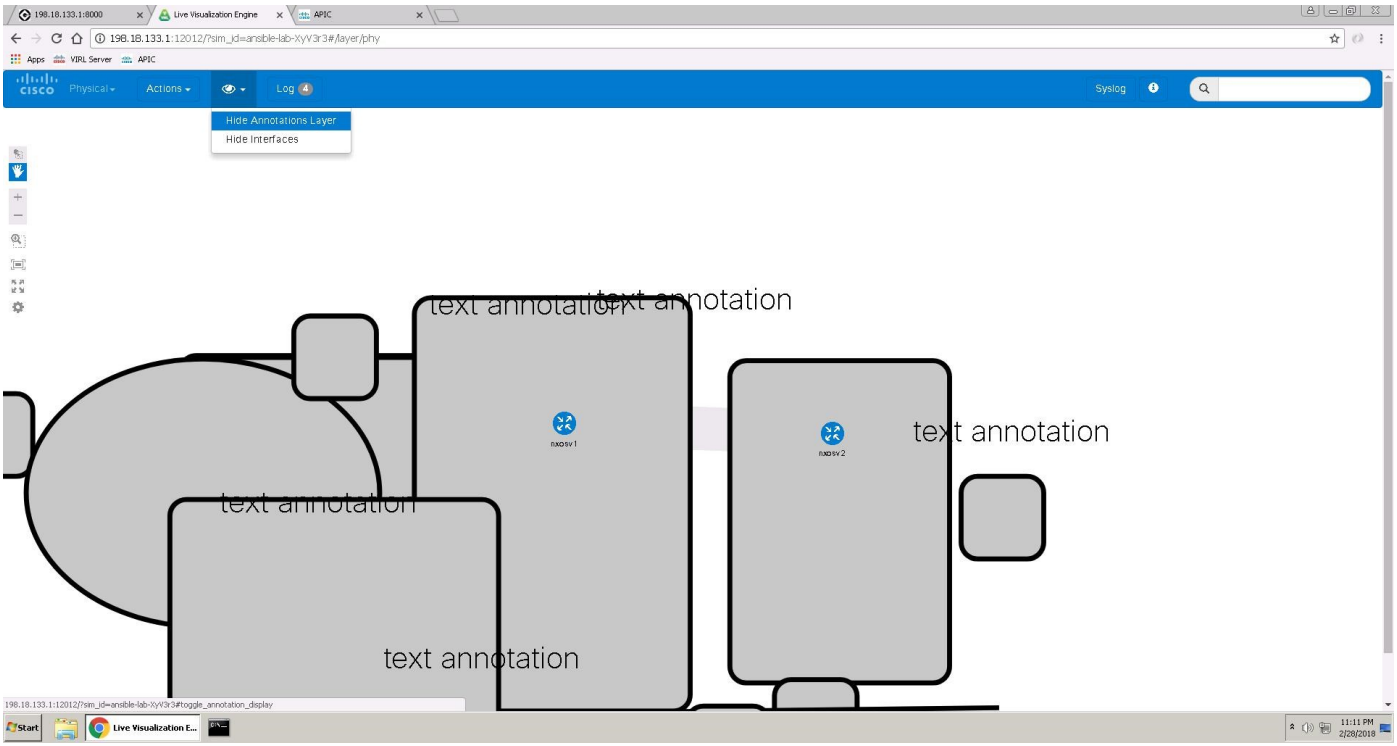
シミュレーション全体（つまり両方の NXOSv デバイス）をリセットする必要がある場合には、このタブの [現在のデモをリセット（Reset my Demo）] ボタンを使用することを推奨します。



注：デモは、リセット後、自動的に開始されます。スタート ボタンを使用する必要はありません。

2 番目のタブには、トポロジ内の問題をデバッグするライブ ビジュアル化ツールがあります。アクティブなシミュレーションがある場合、以下のスクリーン ショットのように、ドロップダウンリストにシミュレーション ID が表示されます。シミュレーションを選択し、[送信（Submit）] ボタンをクリックします。

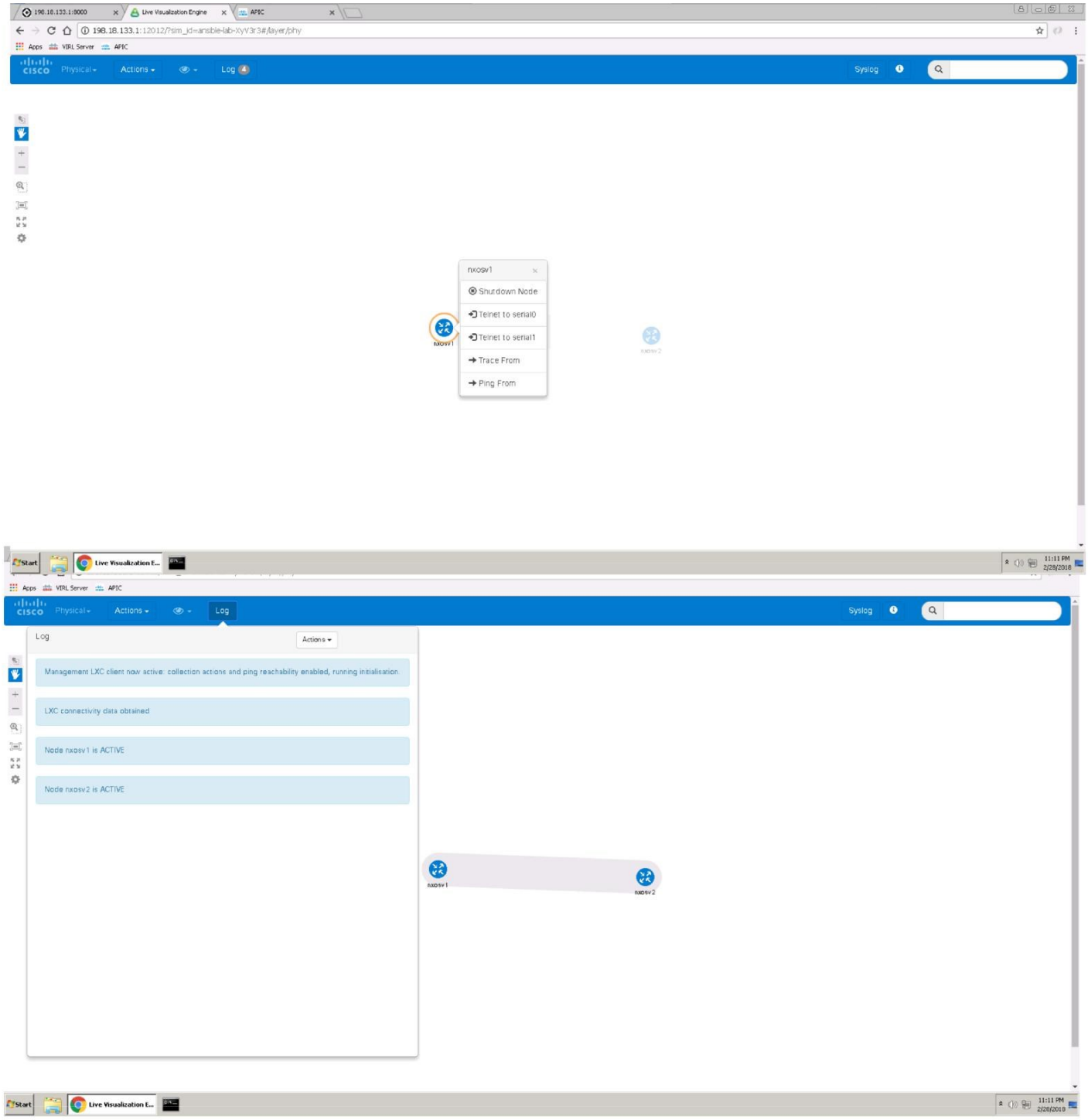
[トポロジライブビジュアル化（topology live visualization）] ウィンドウで目の形のアイコンをクリックしてから、[注釈レイヤを非表示（Hide Annotations Layer）] をクリックして、すべてのテキスト注釈を非表示にし、NXOSv デバイスにフォーカスします。



[ログ（Log）] ボタンをクリックして、シミュレーションのステータスに関連するログ エントリを確認します。nxosv1 ノードのステータスが「ACTIVE」となっている場合、シミュレーション自体はアクティブでもまだデバイスに接続できないことを示しています。

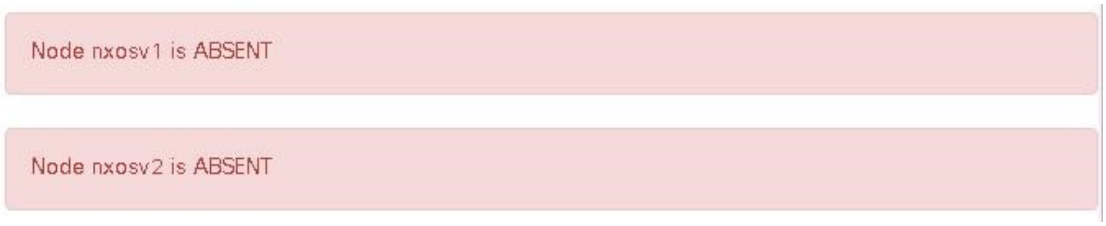
デバイスのアイコンをクリックすると、特定のデバイスへの接続に役立つオプションがいくつか表示されます。

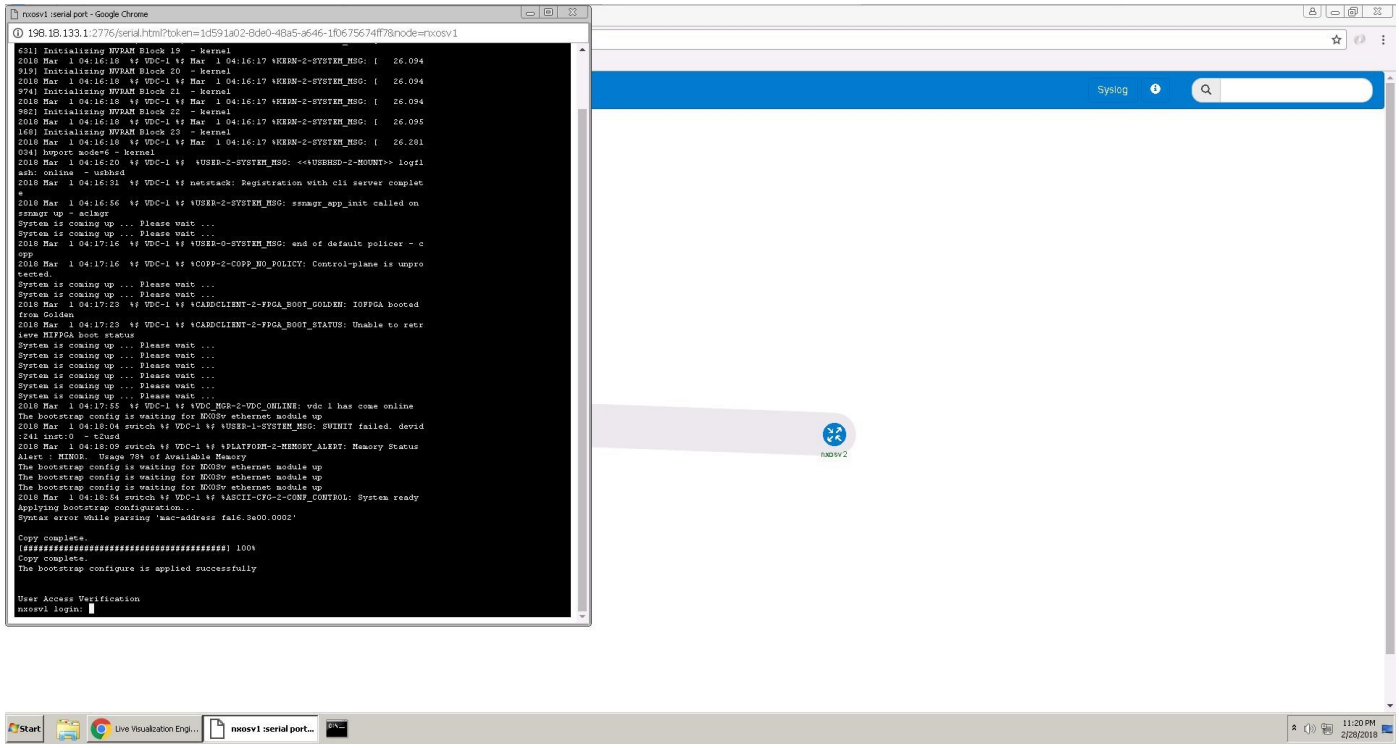
デバイスのいずれかに問題がある場合、または、デバイスをリセットしたい場合は、該当のデバイスをクリックして [ノードのシャットダウン（Shutdown Node）] オプションを選択し、デバイスをシャットダウンします。



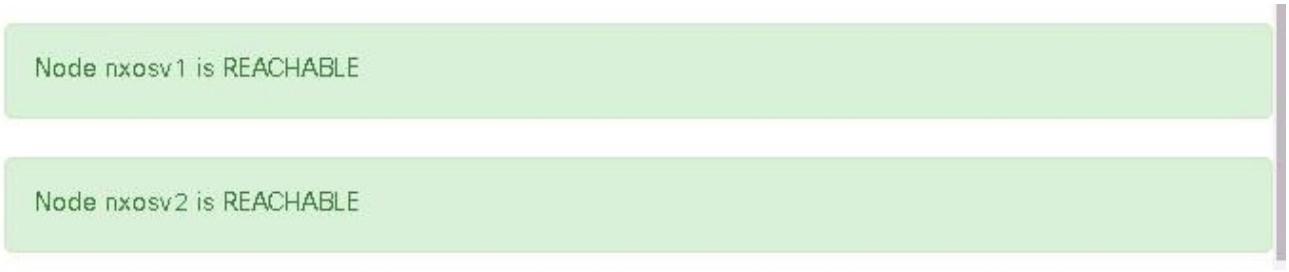
デバイスがシャットダウンされると、ログに表示されるステータスが「ABSENT」になります。

NXOSv ノードを再起動した後、起動して接続可能になるまでに約 5 ~ 10 分かかります。telnet で serial0 に接続すると、各デバイスの起動状態をモニタすることができます。そのためには、トポロジ内のデバイスをクリックし、[Telnet] を選択して serial0 に接続します。Chrome で新しいウィンドウが開き、デバイスへの telnet セッションが表示されます。以下のスクリーンショットのように、デバイスにログインするプロンプトが表示されたら、デバイスは接続可能状態にあります。





デバイスが起動して接続可能になると、ログのステータスは「REACHABLE」に変わります。



シナリオ 1. NXOS の自動化

価値提案：このシナリオでは、NXOS がさまざまな度合いの自動化や信頼性を持つツールから設定することを目的としています。まず Python スクリプトを使用して NXOS を自動化し、その後、Ansible の詳細を確認します。Ansible を利用すると、コンピューティング、ストレージ、ネットワー キングなどインフラストラクチャ内のさまざまな部分にまたがるタスクを、簡単に自動化することができます。ユーザは、このシナリオの 4 つのラボ演習を通して、NXOS と Ansible の利用方法を学びます。まず Ansible の基本的な使い方から始め、その後、高度な使い方に進みます。

ラボ 1. Python を使用した自動化

この演習では、さまざまな自動化や信頼性を持つツールを利用し、いくつかのネットワークの設定方法を参加者に紹介します。必要となる手作業、各方法の信頼性、さまざまなシチュエーションにおけるシームレスかつ拡張可能なネットワーク運用への対応度を比較するために、参加者は、利用可能な自動化オプションについて理解している必要があります。

ネットワーク オペレータの多くは、設定の自動化にシェルまたは Python スクリプトを使用しています。ここでは、下記の Python のサンプルを（そのまま）使用して、ネットワーク デバイスに CLI コマンドを挿入してみます。NXOS 上では、Unix シェルと Python インタープリタのいずれも利用可能です。これにより、ローカルのプログラム実行環境を使用して、スクリプトを作成しスイッチの運用を自動化することができます。Cisco Nexus スイッチには Python 2.7.5 がインストールされているため、プログラムによりスイッチの CLI にアクセスしてさまざまなタスクを実行することができます。

CLI の自動化

1. リモート ワークステーションのデスクトップで、「ansible」ホストに接続する Putty ショートカットを起動します。

2. 次の Python スクリプトを（そのまま）使用して、ネットワーク デバイスに CLI コマンドを挿入してみましょう。ansible の Putty ターミナルで次のコマンドを入力し、VI エディタで Python スクリプトを表示/編集します。

```
[ansible@ansible ~]$ vi dc-automation-bootcamp/nxos/lab1/cli/enable_feature_cli.py
```

スクリプト内のコメントは、各コマンドの内容を説明しています。スクリプト全体を確認してその作用を理解してください。

```
#!/usr/bin/env python2

import paramiko # SSH セッションで利用
import time # スクリプト内で一時停止をするために利用

# アクセスするホストの一覧
hosts = ["nxosv1"]

# NXOS login 情報
username = "ansible"
password = "ansible"

# SSH 用に新しい Paramiko の接続を作成
object conn = paramiko.SSHClient()
# 自動的にホストの SSH 鍵を登録
conn.set_missing_host_key_policy(paramiko.AutoAddPolicy())
```

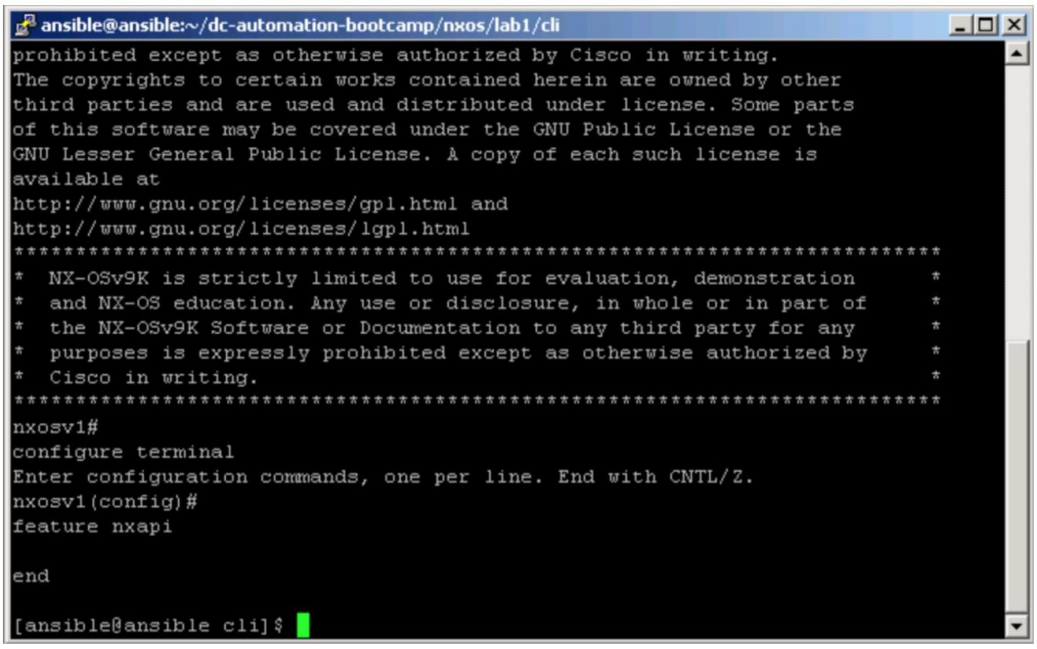
```
# 接続対象のホストすべてに対して実行する
for host in hosts:
        print "------------------- ",host,"-------------------- "
        # 複数コマンドを実行するために shell のセッションを作成する
        conn.connect(host, 22, username, password, look_for_keys=False, allow_agent=False)
        remote_shell = conn.invoke_shell()
        time.sleep(2)
        # リモートホストの出力を受信
        output = remote_shell.recv(65535)
        # 出力を表示
        print output

        # "configure terminal" を実行
        remote_shell.send("configure terminal\n")
        time.sleep(1)
        output = remote_shell.recv(65535)
        print output

        # NXAPI 機能を有効にする
        remote_shell.send("feature nxapi\n")
        time.sleep(1)
        output = remote_shell.recv(65535)
        print output

        # コンフィギュレーション モードを抜
        ける remote_shell.send("end\n")
        time.sleep(1)
        output = remote_shell.recv(65535)
        print output
        time.sleep(1)

        # SSH のセッションを切断し、同一 SSH クライアントを再利用できるようにする
        conn.close()
```

3. スクリプトをひととおり確認したら、VI エディタを終了します。

4. Putty ターミナルで以下のコマンドを入力して、ラボ演習を実行します。

```
[ansible@ansible ~]$ python dc-automation-bootcamp/nxos/lab1/cli/enable_feature_cli.py
```

5. 実行後、端末には次のように表示されます。



出力結果

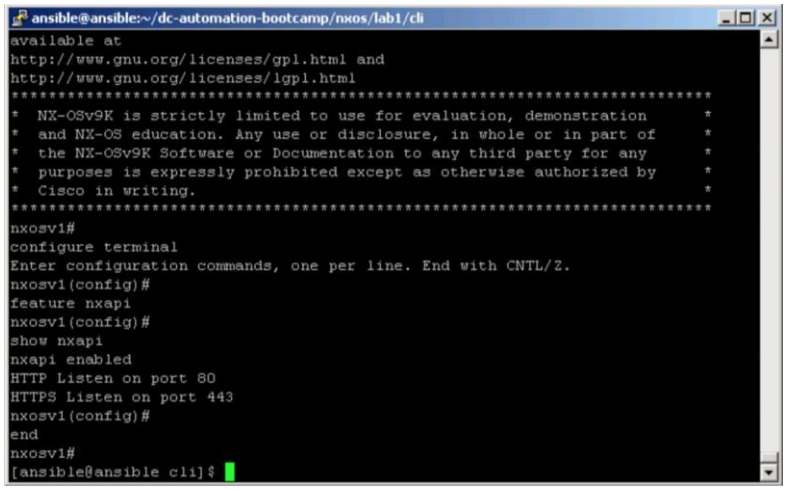
```
[ansible@ansible ~]$ python dc-automation-bootcamp/nxos/lab1/cli/enable_feature_cli.py
-------------------- nxosv1 --------------------

Cisco NX-OS Software
Copyright (c) 2002-2017, Cisco Systems, Inc. All rights reserved.
NX-OSv9K software ("NX-OSv9K Software") and related documentation,
files or other reference materials ("Documentation") are
the proprietary property and confidential information of Cisco
Systems, Inc. ("Cisco") and are protected, without limitation,
pursuant to United States and International copyright and trademark
laws in the applicable jurisdiction which provide civil and criminal
penalties for copying or distribution without Cisco's authorization.

Any use or disclosure, in whole or in part, of the NX-OSv9K Software
or Documentation to any third party for any purposes is expressly
prohibited except as otherwise authorized by Cisco in writing.
The copyrights to certain works contained herein are owned by other
third parties and are used and distributed under license. Some parts
of this software may be covered under the GNU Public License or the
GNU Lesser General Public License. A copy of each such license is
available at
http://www.gnu.org/licenses/gpl.html and
http://www.gnu.org/licenses/lgpl.html
*****************************************************************************
* NX-OSv9K is strictly limited to use for evaluation, demonstration        *
* and NX-OS education. Any use or disclosure, in whole or in part of       *
* the NX-OSv9K Software or Documentation to any third party for any        *
*  purposes is expressly prohibited except as otherwise authorized by      *
*  Cisco in writing.                                                       *
*****************************************************************************
```

```
nxosv1#
configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
nxosv1(config)#
feature nxapi

end
```

6. Python のソース ファイルに、NXAPI 機能のステータスを表示するコード ブロックを追加します。VI エディタでスクリプトを開き（ステップ 2 を参照）、下記の赤字部分のコード ブロックを追加します。変更を保存し、VI エディタを終了します。

```
        # NXAPI を有効にする
        remote_shell.send("feature nxapi\n")
        time.sleep(1)
        output = remote_shell.recv(65535)
        print output

        # Show the NXAPI feature status
        remote_shell.send("show nxapi\n")
        time.sleep(1)
        output = remote_shell.recv(65535)
        print output
        time.sleep(1)

        # コンフィギュレーション モードを抜ける
        remote_shell.send("end\n")
        time.sleep(1)
        output = remote_shell.recv(65535)
        print output
        time.sleep(1)
```

7. 編集したスクリプトを実行します（ステップ 4 を参照）。次の出力が表示されます。



NXAPI の自動化

Cisco Nexus デバイスでは、設定はコマンドライン インターフェイス（CLI）を使用して実施します。CLI は、当該デバイス上でしか実行できません。Nexus の設定をしやすくするために、シスコは、HTTP/HTTPS API を提供することにより NX-API REST を導入して、以下のことを可能にしています。

特定の CLI コマンドをスイッチの外部から実行可能。

少ない HTTP/HTTPS 操作上で複数の設定アクションを実行可能。

NX-API REST は、コマンドやスイッチへの設定に対応しているだけでなく、Linux Bash コマンドも実行可能。

NX-API REST は、転送に HTTP または HTTPS を使用します。設定コマンドは HTTP/HTTPS の body 部に組み込まれ、配信メソッドとして POST が使用されます。NX-API REST は、裏で Nginx HTTP サーバを使用しています。Nginx サーバに対しては、要求したデータを XML または JSON 形式で取得可能です。

注：NXAPI を使用するには、スイッチで *nxapi* 機能が有効になっていることを確認してください。

NXAPI は、モデル駆動型のアプローチを採用しています。各モデルのオブジェクトには、一意の URL からアクセスできます。詳細については、https://developer.cisco.com/docs/nxapi-dme-model-reference/ [英語] を参照してください。

1. Postman などの REST API クライアントを使用して NXAPI コールを実行したり、Python スクリプトにラップして設定タスクを自動化したりすることができます。この演習では、Python スクリプトにラップされた NXAPI を使用して、スイッチ上で *interface-vlan* 機能を有効にします。ansible の Putty ターミナルで次のコマンドを入力し、VI エディタで Python スクリプトを表示/編集します。

```
[ansible@ansible ~]$ vi dc-automation-bootcamp/nxos/lab1/rest/enable_feature_nxapi.py
```

スクリプト内のコメントは、各コマンドの内容を説明しています。スクリプト全体を確認してその作用を理解してください。

```
import requests # HTTP リクエストに利用
import urllib3 # HTTP リクエストに利用
import json # JSON を解析するのに利用

urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

# アクセスするホストの一覧
hosts = ["nxosv1"]

# 接続対象のホストすべてに対して実行する
for host in hosts:
    print "\nAuthenticating with the device . . .\n"
    # 認証トークンを取得
    url = "https://" + host + "/api/mo/aaaLogin.json"
    # Login data
    data = """
    {
      "aaaUser":
        { "attributes":
        {
          "name": "ansible",
          "pwd": "ansible"
        }
      }
    }
    """
    response = requests.post(url, data=data, headers={'Content-Type': 'application/json'}, verify=False)
    # JSON をそのまま出力する場合、下記行をコメントアウトする
    # print json.dumps(response.json(), indent=2)

    if response.status_code == requests.codes.ok:
        print "Authentication successful!\n"
    else:
        print "Authentication failed! Please verify login credentials!\n"
        exit(0)
```

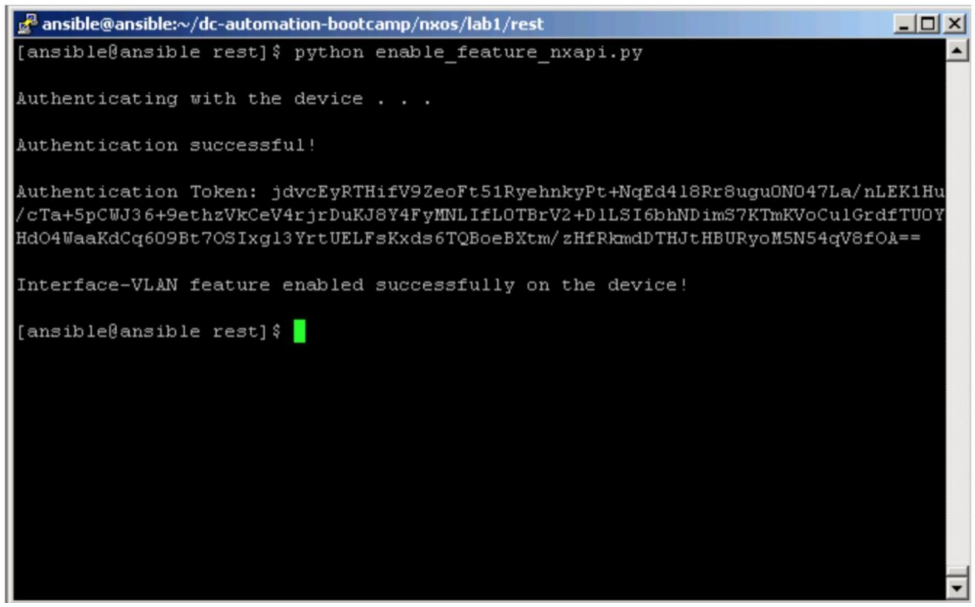
```
    token = response.json()['imdata'][0]['aaaLogin']['attributes']['token']
    print "Authentication Token: " + token + "\n"

    # Interface-Vlan 機能を有効にする
    url = "https://" + host + "/api/mo/sys.json"

    data = """{
      "topSystem": {
        "children": [
          {
            "fmEntity": {
              "children": [
                {
                  "fmInterfaceVlan": {
                    "attributes": {
                      "adminSt": "enabled"
                    }
                  }
                }
              ]
            }
          }
        ]
      }
    }"""

    cookie = {'APIC-Cookie': token}
    response = requests.post(url, cookies=cookie, data=data, headers={'Content-Type':
'application/json'}, verify=False)

    if response.status_code == requests.codes.ok:
        print "Interface-VLAN feature enabled successfully on the device!\n"
    else:
        print "ERROR: Could not enable Interface-VLAN feature!\n"
        exit(0)
```

2. スクリプトをひととおり確認したら、VI エディタを終了します。

3. Putty ターミナルで以下のコマンドを入力して、ラボ演習を実行します。

```
[ansible@ansible ~]$ python dc-automation-bootcamp/nxos/lab1/rest/enable_feature_nxapi.py
```

実行後、端末には次のように表示されます。



出力結果

```
[ansible@ansible ~]$ python dc-automation-bootcamp/nxos/lab1/rest/enable_feature_nxapi.py

Authenticating with the device . . .

Authentication successful!

Authentication Token:
MtoUYXhIeIYZ5QMbH3SQ96Jv6cJagbB17jVCnoMR+NX4fYoiuXEiGn0jDpj22aGghd7OZe2qreGZ+dGbLVSFldclUdsGMo7OhFCZ71rt
l2P10OaZ+FlNKeW+w6mcwzt256HmGwE5VUYGnz+dEo+e/woQjTuO0myrVFe48rQzLOK1s3A3Owfld0iqYgbd/wnG9r4Wz24Cxalwmmnc
k15ixw==

Interface-VLAN feature enabled successfully on the device!
```

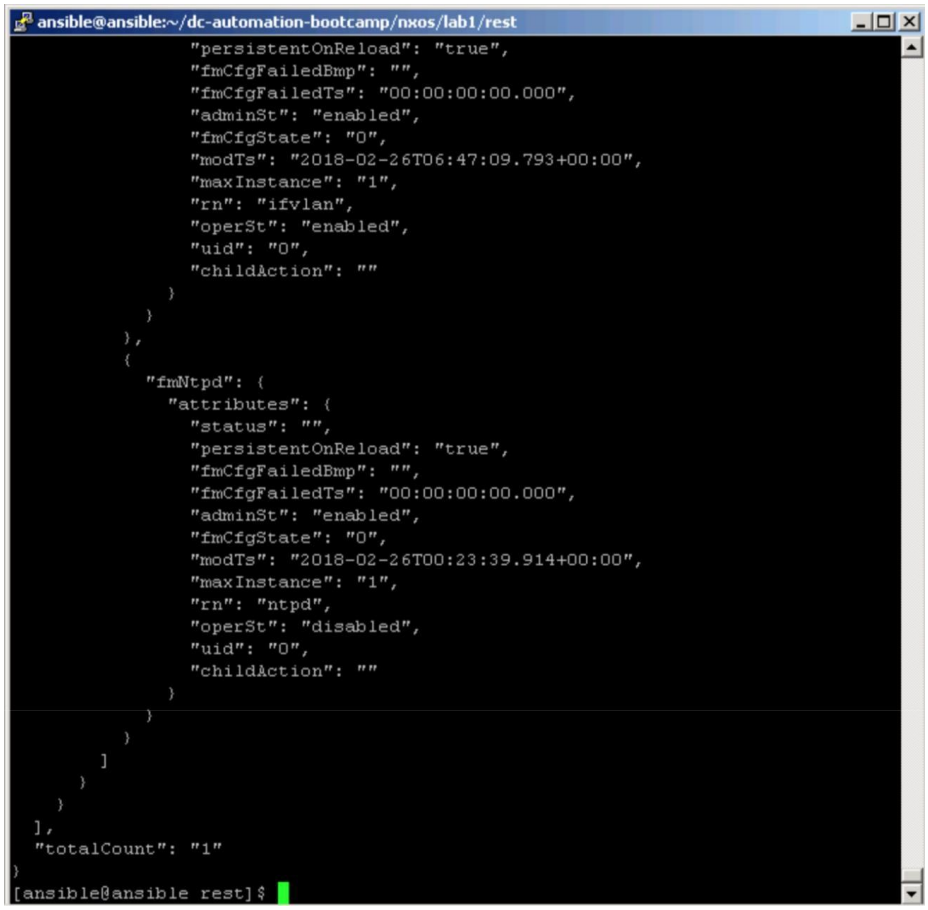
4. Python のソース ファイルに、デバイスで有効な機能のリストを取得するためのコード ブロックを追加します。それを行うために、デバイスに GET リクエストを送信します。VI エディタでスクリプトを開き（ステップ 2 を参照）、下記の赤字部分のコード ブロックを追加します。変更を保存し、VI エディタを終了します。

```
    token = response.json()['imdata'][0]['aaaLogin']['attributes']['token']
    print "Authentication Token: " + token + "\n"

    # Enable Interface-Vlan feature on the device
    url = "https://" + host + "/api/mo/sys/fm.json?rsp-subtree=children"
    cookie = {'APIC-Cookie': token}

    response = requests.get(url, cookies=cookie, headers={'Content-Type': 'application/json'},
verify=False)

    if response.status_code == requests.codes.ok:
        print json.dumps(response.json(), indent=2)
    else:
        print "ERROR: Could not fetch the system information!\n"
        exit(0)
```

「sys/fm」は、「GET」リクエストで取得する機能モデル オブジェクトへのパスです。各機能は「sys/fm」の下にあるので、リクエストに ?rsp-subtree=children を含めて、NXOS が応答の中に子階層の MO を含められるようにしています。

5. 編集したスクリプトを実行します（ステップ 3 を参照）。次の出力が表示されます。

   NXAPI 機能がどこに表示されているか、また、その有効/無効の別をどうやって判断できるか、おわかりでしょうか。

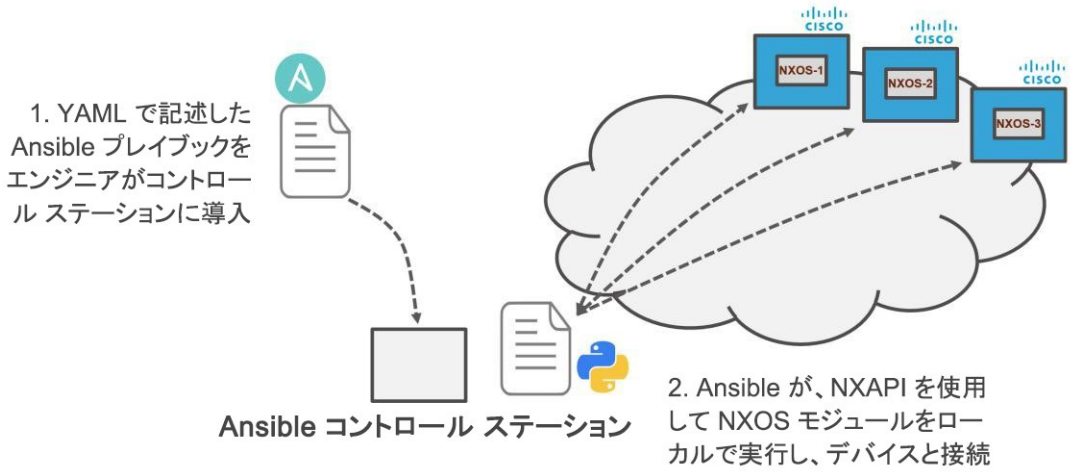


これでシナリオ 1 のラボ 1 は完了です。

ラボ 2. Ansible の基礎

このモジュールでは、Ansible を利用して NX-OS を自動化するための基礎を確認します。Ansible を利用すると、コンピューティング、ストレージ、ネットワー キングなどインフラストラクチャ内のさまざまな部分にまたがるタスクを、簡単に自動化することができます。Ansible を利用した一般的なサーバ運用では、Python コードが管理対象上で実行されますが、Cisco NX-OS デバイスで Ansible を実行する場合は、NXAPI が使用されます。このような一元運用モデルは、大規模ネットワークのオペレータにとってメリットが大きく、それが、サービス プロバイダー業界において Ansible が人気を得ている理由の 1 つになっています。

図 4. Ansible と Nx-OS の連携の仕組み



注：NXOS の各モジュールに関する詳細情報については、次のリンクを参照してください - http://docs.ansible.com/ansible/latest/list_of_network_modules.html#nxos [英語]。各モジュールが詳細に説明され、設定可能なすべてのパラメータや戻り値の例が記載されています。

インベントリ ファイルには、Ansible で操作するホストが含まれています。また、Ansible プレイブックから参照できるグループとホストの定義も含まれます。VI エディタを使用してインベントリ ファイルを確認するには、次のコマンドを入力します。

```
[ansible@ansible ~]$ vi dc-automation-bootcamp/nxos/lab2/inventory
```

```
nxosv1

[all:vars]
ansible_connection = local
ansible_ssh_user = ansible
ansible_ssh_pass = ansible
```

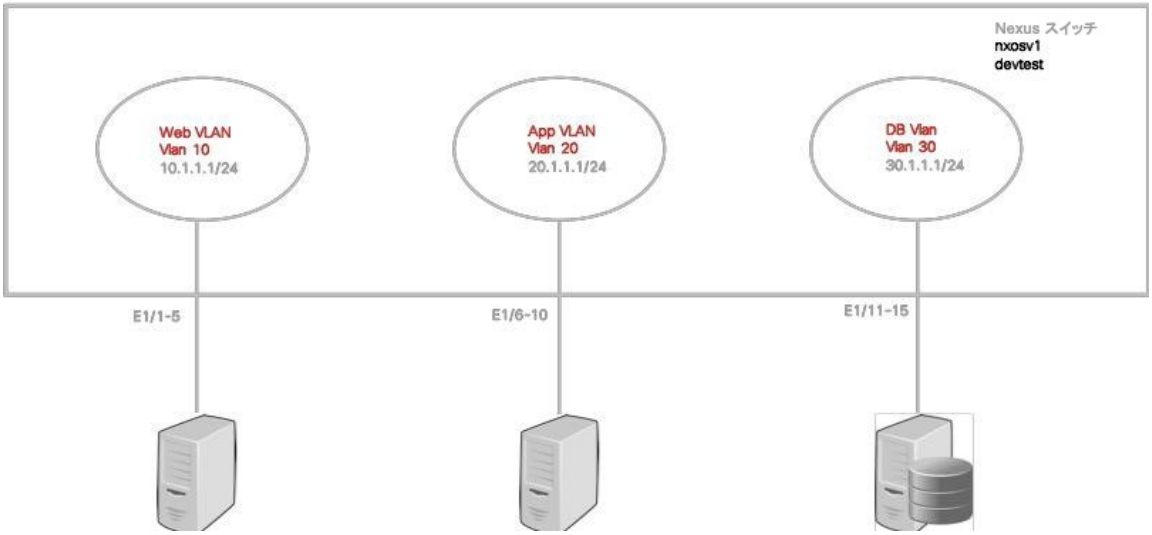
このラボ演習では、***nxos_vlan*** モジュールを使用して、VLAN を作成するためのシンプルなプレイブックを作成/実行します。以下を作成します。

Web VLAN – 10

App VLAN – 20

DB VLAN – 30

図 5. VLAN の作成



ここでは、設定値を Ansible タスクに渡すための方法を 3 つ紹介します。タスクに値を直接記述する方法、プレイブックに定義した変数を使用する方法、外部ファイルに定義した変数を使用する方法の 3 つです。

1. ansible ホストの Putty ターミナルで lab2 ディレクトリに切り替えます。

```
[ansible@ansible ~]$ cd ~/dc-automation-bootcamp/nxos/lab2/
```

2. ansible の Putty ターミナルで次のコマンドを入力し、VI エディタで ansible プレイブックを表示/編集します。

```
[ansible@ansible lab2]$ vi create_vlans.yml
```

プレイブック内のコメントは、各セクションの内容を説明しています。プレイブック全体を確認してその作用を理解してください。

注：使用する Ansible モジュール

nxos_vlan - http://docs.ansible.com/ansible/latest/nxos_vlan_module.html [英語]

```
---
- name: VLAN creation Playbook
  hosts: all                          # inventory file 上に定義されたすべてのホストに対してタスクを実行する
  gather_facts: no

  vars:                               # この playbook 上で利用するパラメータを定義する
    vlan20:
      id: 20
      name: app

  vars_files:                         # 外部ファイルからパラメータをインポートする
    - external_vars.yml

  tasks:
  - name: Ensure VLAN 10 exists       # スクリプトに直接書かれた値
    nxos_vlan:
      vlan_id: 10
      name: web
      transport: nxapi
```

```
- name: Ensure VLAN 20 exists      # playbook 内の vars で定義された値
  nxos vlan:
    vlan_id: "{{ vlan20.id }}"
    name: "{{ vlan20.name }}"
    transport: nxapi

- name: Ensure VLAN 30 exists      # 外部ファイル "external_vars.yml" の中で定義されている値。
  nxos_vlan:
    vlan_id: "{{ vlan30.id }}"
    name: "{{ vlan30.name }}"
    transport: nxapi
```

3. プレイブックをひととおり確認したら、VI エディタを終了します。

4. 次に、外部変数ファイルを確認し、外部ファイルで変数がどのように定義されているか理解してください。次のコマンドを入力して、VI エディタで外部変数ファイルを開きます。

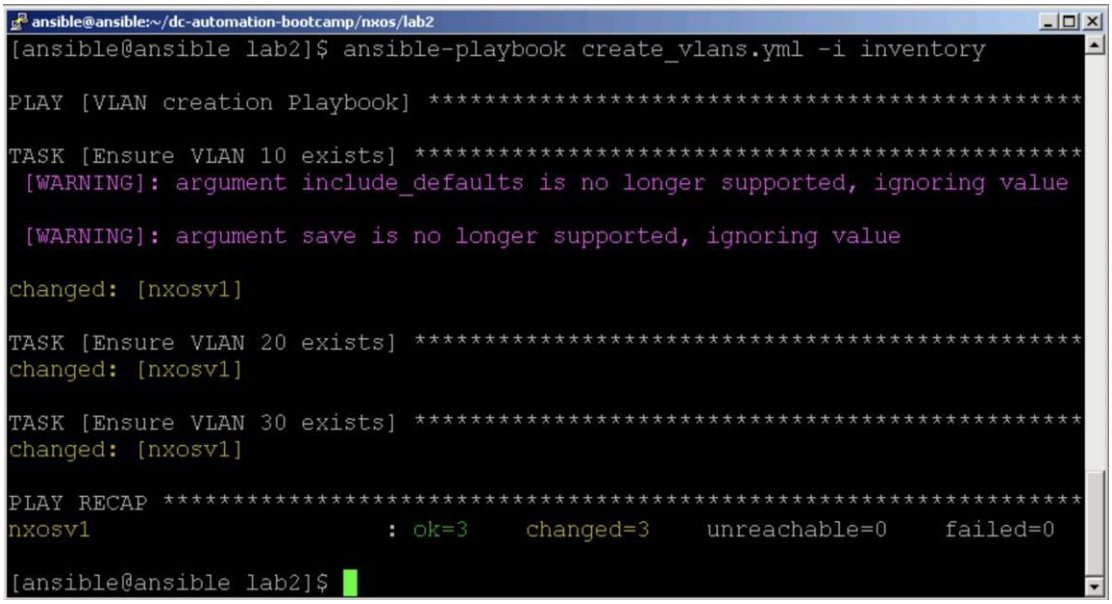
```
[ansible@ansible lab2]$ vi external_vars.yml

---

vlan30:
  id: 30
  name: db
```

5. 外部変数ファイルをひととおり確認したら、VI エディタを終了します。

6. Putty ターミナルで以下のコマンドを入力して、ansible プレイブックを実行します。

```
[ansible@ansible lab2]$ ansible-playbook create_vlans.yml -i inventory
```

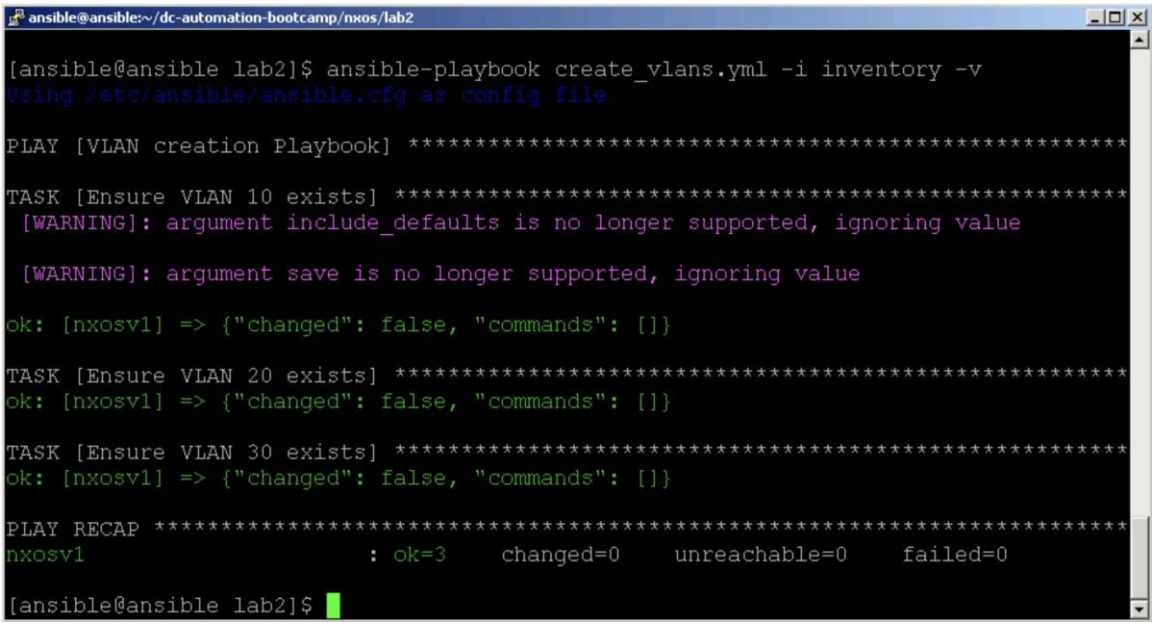
実行後、端末には次のように表示されます。



出力結果

```
[ansible@ansible lab2]$ ansible-playbook create_vlans.yml -i inventory

PLAY [VLAN creation Playbook] ***************************************************************

TASK [Ensure VLAN 10 exists] ****************************************************************
 [WARNING]: argument include_defaults is no longer supported, ignoring value
 [WARNING]: argument save is no longer supported, ignoring value
changed: [nxosv1]

TASK [Ensure VLAN 20 exists] ****************************************************************
changed: [nxosv1]

TASK [Ensure VLAN 30 exists] ****************************************************************
changed: [nxosv1]

PLAY RECAP **********************************************************************************
nxosv1                     : ok=3    changed=3    unreachable=0    failed=0
```

7. オプションを -v、-vv、-vvv に変えながらプレイブックを複数回実行します（ステップ 6 参照）。-v オプションでプレイブックを実行すると、出力結果がより詳細になります。プレイブック実行時に起動されるバックグラウンド プロセスに関する情報が、より多く表示されます。「v」を追加するごとに、情報がより詳細になります。



これでシナリオ 1 のラボ 2 は完了です。

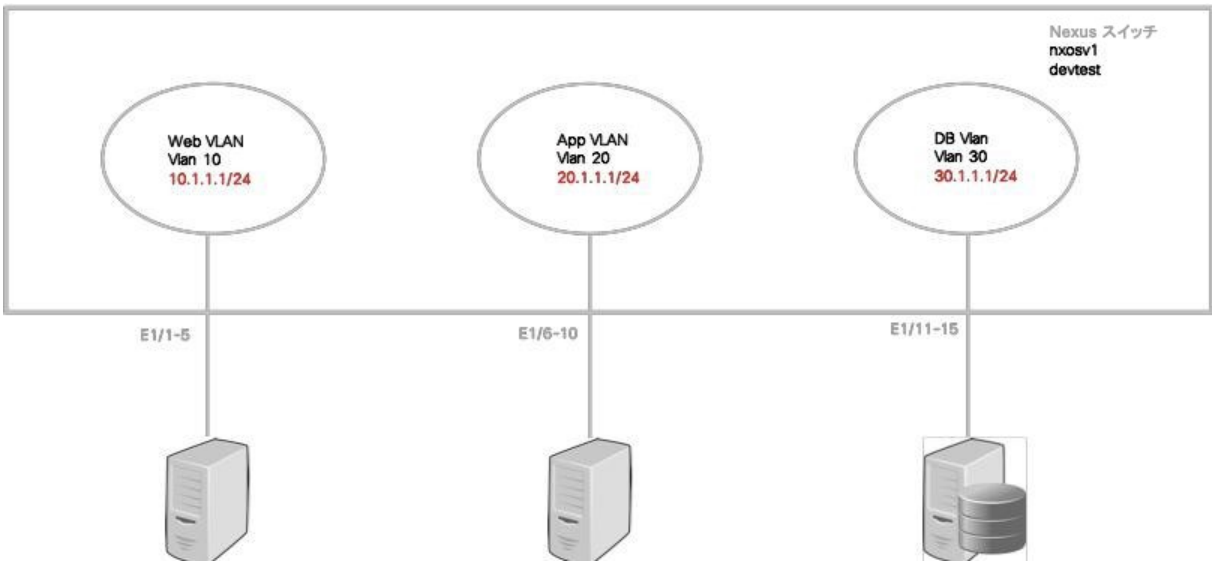
ラボ 3. Ansible の基本的用法

このラボ演習では、Ansible で*条件文*と*ループ*を使用する方法を学びます。条件文を使用すると、実行対象のタスクやその実行方法をより細かく制御することができます。ループを使用すると、一連のデバイスや設定値に対し、プログラムによってタスクを繰り返し実行することができ、自動化スクリプトを再利用しやすくなります。最後に、テンプレートを使用して定型書式の出力ファイルを生成する方法を確認します。

3.1 ループ

複数の VLAN インターフェイスを設定してみましょう。各インターフェイスを設定するために同じタスクを逐一記述するのではなく、Ansible のループ ステートメント（*with_items*）を使用して、複数のインターフェイスに対して同じタスクを繰り返し実行します。複数の VLAN インターフェイスそれぞれに IPv4 アドレスを設定します。

図 6. 各 VLAN インターフェイスの IPv4 アドレス設定



1. ansible ホストの Putty ターミナルで 3.1.loops ディレクトリに切り替えます。

```
[ansible@ansible ~]$ cd ~/dc-automation-bootcamp/nxos/lab3/3.1.loops
```

2. 外部変数ファイルを確認し、変数がどのように定義されているか理解してください。次のコマンドを入力して、VI エディタで外部変数ファイルを開きます。

```
[ansible@ansible 3.1.loops]$ vi external_vars.yml

---

vlan_interfaces:
  - { name: Vlan10, desc: Web VLAN, ipv4: 10.1.1.1, mask: 24 }
  - { name: Vlan20, desc: App VLAN, ipv4: 20.1.1.1, mask: 24 }
  - { name: Vlan30, desc: DB VLAN, ipv4: 30.1.1.1, mask: 24 }
```

3 つの VLAN に対する設定データがすべて変数ファイルに保存されています。vlan_interfaces 変数には 3 つの項目が含まれており、各項目に、name、desc、ipv4、mask 変数が定義されています。

3. ansible の Putty ターミナルで次のコマンドを入力し、VI エディタで ansible プレイブックを表示/編集します。

```
[ansible@ansible 3.1.loops]$ vi config_vlan_intf.yml
```

プレイブック内のコメントは、各セクションの内容を説明しています。プレイブック全体を確認してその作用を理解してください。

注：使用する Ansible モジュール

nxos_interface - http://docs.ansible.com/ansible/latest/nxos_interface_module.html [英語]

nxos_ip_interface - http://docs.ansible.com/ansible/latest/nxos_ip_interface_module.html [英語]

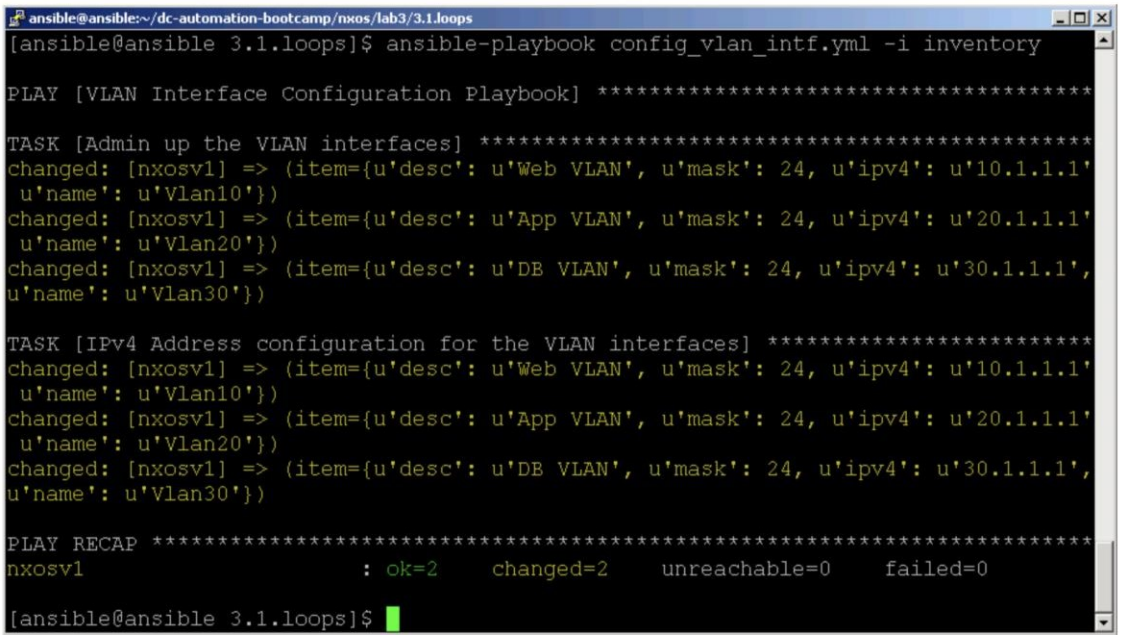
```
---
- name: VLAN Interface Configuration Playbook
  hosts: all
  gather_facts: no
  vars_files:
    - external_vars.yml

  tasks:
  - name: Admin up the VLAN interfaces # The task will execute for each item defined under
vlan_interfaces variable
    nxos_interface:
      interface: "{{ item.name }}"
      description: "{{ item.desc }}"
      admin_state: up
      transport: nxapi
    with_items:
      "{{ vlan_interfaces }}"

  - name: IPv4 Address configuration for the VLAN
    interfaces nxos_ip_interface:
      interface: "{{ item.name }}"
      version: v4
      addr: "{{ item.ipv4 }}"
      mask: "{{ item.mask }}"
      transport:
    nxapi with_items:
      "{{ vlan_interfaces }}"
```

4. プレイブックをひととおり確認したら、VI エディタを終了します。

5. Putty ターミナルで以下のコマンドを入力して、ansible プレイブックを実行します。

```
[ansible@ansible 3.1.loops]$ ansible-playbook config_vlan_intf.yml -i inventory
```

実行後、端末には次のように表示されます。



出力結果

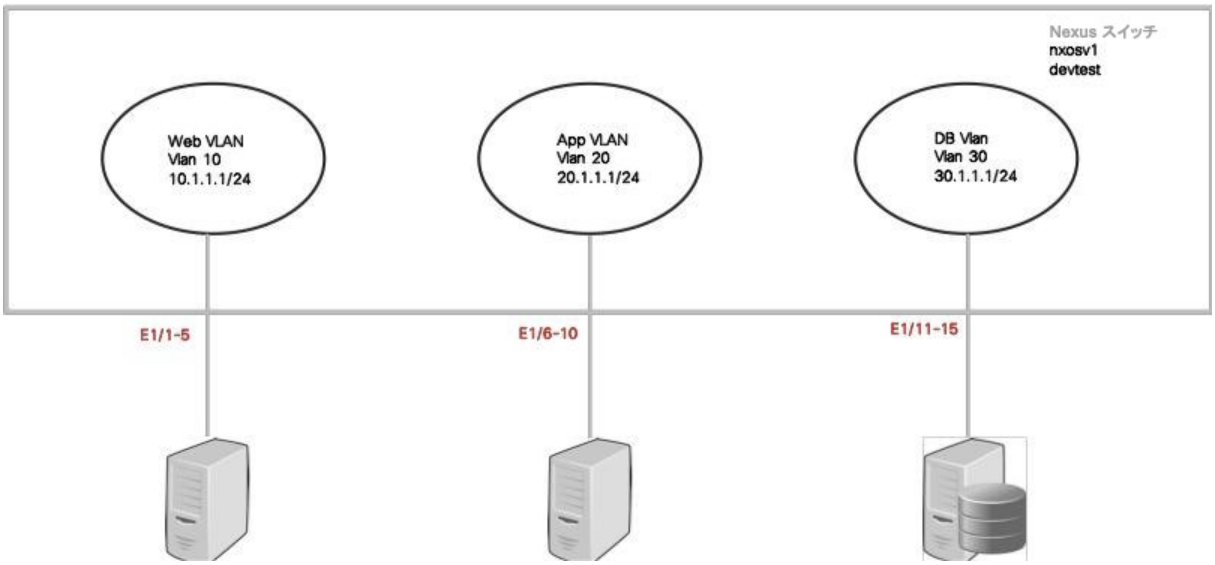
```
[ansible@ansible 3.1.loops]$ ansible-playbook config_vlan_intf.yml -i inventory

PLAY [VLAN Interface Configuration Playbook] ************************************************************

TASK [Admin up the VLAN interfaces] *********************************************************************
changed: [nxosv1] => (item={u'desc': u'Web VLAN', u'mask': 24, u'ipv4': u'10.1.1.1', u'name': u'Vlan10'})
changed: [nxosv1] => (item={u'desc': u'App VLAN', u'mask': 24, u'ipv4': u'20.1.1.1', u'name': u'Vlan20'})
changed: [nxosv1] => (item={u'desc': u'DB VLAN', u'mask': 24, u'ipv4': u'30.1.1.1', u'name': u'Vlan30'})

TASK [IPv4 Address configuration for the VLAN interfaces] ***********************************************
changed: [nxosv1] => (item={u'desc': u'Web VLAN', u'mask': 24, u'ipv4': u'10.1.1.1', u'name': u'Vlan10'})
changed: [nxosv1] => (item={u'desc': u'App VLAN', u'mask': 24, u'ipv4': u'20.1.1.1', u'name': u'Vlan20'})
changed: [nxosv1] => (item={u'desc': u'DB VLAN', u'mask': 24, u'ipv4': u'30.1.1.1', u'name': u'Vlan30'})

PLAY RECAP **********************************************************************************************
nxosv1                     : ok=2    changed=2    unreachable=0    failed=0
```

3.2 条件文

このラボ演習では、条件文の使用方法を確認します。条件文を使用することで、プレイブックでの実行対象となるタスクを、より細かく制御することができます。特に、条件文をループと合わせて使用すると便利です。

ここでは、物理インターフェイスに関連付けられている VLAN に基づいて、物理インターフェイスの記述を設定します。インターフェイス Ethernet1-5 は Web VLAN（10）、Ethernet6-10 は App VLAN（20）、Ethernet11-15 は DB VLAN（30）に、それぞれ関連付けられています。

図 7. 物理インターフェイスの設定



1. ansible ホストの Putty ターミナルで 3.2.conditionals ディレクトリに切り替えます。

```
[ansible@ansible ~]$ cd ~/dc-automation-bootcamp/nxos/lab3/3.2.conditionals
```

2. 外部変数ファイルを確認し、変数がどのように定義されているか理解してください。次のコマンドを入力して、VI エディタで外部変数ファイルを開きます。

```
[ansible@ansible 3.2.conditionals]$ vi external_vars.yml

---

physical_interfaces:
  - { name: Ethernet1/1 , vlan: 10 }
  - { name: Ethernet1/2 , vlan: 10 }
  - { name: Ethernet1/3 , vlan: 10 }
  - { name: Ethernet1/4 , vlan: 10 }
  - { name: Ethernet1/5 , vlan: 10 }
  - { name: Ethernet1/6 , vlan: 20 }
  - { name: Ethernet1/7 , vlan: 20 }
  - { name: Ethernet1/8 , vlan: 20 }
  - { name: Ethernet1/9 , vlan: 20 }
  - { name: Ethernet1/10 , vlan: 20 }
  - { name: Ethernet1/11 , vlan: 30 }
  - { name: Ethernet1/12 , vlan: 30 }
  - { name: Ethernet1/13 , vlan: 30 }
  - { name: Ethernet1/14 , vlan: 30 }
  - { name: Ethernet1/15 , vlan: 30 }
```

これから、変数ファイル内でインターフェイスに定義されている VLAN に基づいて Web VLAN、App VLAN、DB VLAN のいずれかにインターフェイス記述を設定するタスクを、プレイブックに書き込んでいきます。たとえば、Web インターフェイスを設定するタスクの場合、各物理インターフェイスの *vlan* 変数が 10 であるかどうかチェックします。

3. ansible の Putty ターミナルで次のコマンドを入力し、VI エディタで ansible プレイブックを表示/編集します。

```
[ansible@ansible 3.2.conditionals]$ vi set_interface_descr.yml
```

プレイブック内のコメントは、各セクションの内容を説明しています。プレイブック全体を確認してその作用を理解してください。

注：使用する Ansible モジュール

nxos_interface - http://docs.ansible.com/ansible/latest/nxos_interface_module.html [英語]

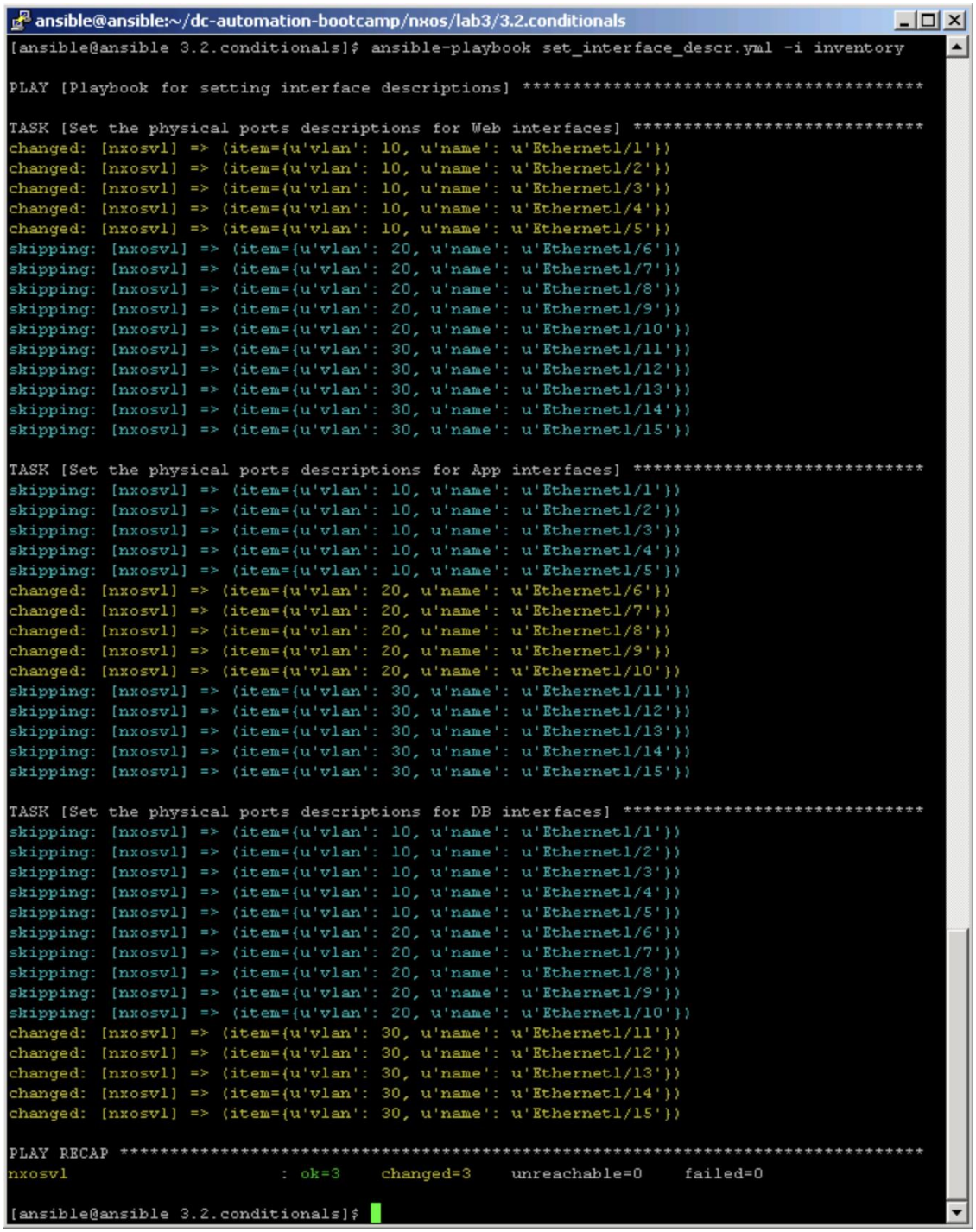
```
---
- name: Playbook for setting interface descriptions
  hosts: all
  gather_facts: no
  vars_files:
    - external_vars.yml

  tasks:
  - name: Set the physical ports descriptions for Web
    interfaces nxos_interface:
      interface: "{{ item.name }}"
      description: Web Vlan
      transport: nxapi
    with_items:                          # このタスクは、pyhsical interfaces 配下の要素すべてに対して実行されます。
      "{{ physical_interfaces }}"        # vlan の値が 10 の要素だけ抽出されます。
    when:
      - item.vlan == 10

  - name: Set the physical ports descriptions for App
    interfaces nxos_interface:
      interface: "{{ item.name }}"
      description: App Vlan
      transport:
    nxapi with_items:
      "{{ physical_interfaces }}
    " when:
      - item.vlan == 20

  - name: Set the physical ports descriptions for DB
    interfaces nxos_interface:
      interface: "{{ item.name }}"
      description: DB Vlan
      transport:
    nxapi with_items:
      "{{ physical_interfaces }}
    " when:
      - item.vlan == 30
```

4. プレイブックをひととおり確認したら、VI エディタを終了します。

5. Putty ターミナルで以下のコマンドを入力して、ansible プレイブックを実行します。

```
[ansible@ansible 3.2.conditionals]$ ansible-playbook set_interface_descr.yml -i inventory
```

実行後、端末には次のように表示されます。



出力結果

```
[ansible@ansible 3.2.conditionals]$ ansible-playbook set_interface_descr.yml -i inventory
```

```
PLAY [Playbook for setting interface descriptions] ******************************************************

TASK [Set the physical ports descriptions for Web interfaces] ***********************************************
changed: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/1'})
changed: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/2'})
changed: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/3'})
changed: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/4'})
changed: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/5'})
skipping: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/6'})
skipping: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/7'})
skipping: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/8'})
skipping: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/9'})
skipping: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/10'})
skipping: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/11'})
skipping: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/12'})
skipping: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/13'})
skipping: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/14'})
skipping: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/15'})

TASK [Set the physical ports descriptions for App interfaces] ***********************************************
skipping: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/1'})
skipping: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/2'})
skipping: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/3'})
skipping: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/4'})
skipping: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/5'})
changed: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/6'})
changed: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/7'})
changed: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/8'})
changed: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/9'})
changed: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/10'})
skipping: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/11'})
skipping: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/12'})
skipping: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/13'})
skipping: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/14'})
skipping: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/15'})

TASK [Set the physical ports descriptions for DB interfaces] ************************************************
skipping: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/1'})
skipping: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/2'})
skipping: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/3'})
skipping: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/4'})
skipping: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/5'})
skipping: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/6'})
skipping: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/7'})
skipping: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/8'})
skipping: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/9'})
skipping: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/10'})
changed: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/11'})
changed: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/12'})
changed: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/13'})
changed: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/14'})
changed: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/15'})

PLAY RECAP **************************************************************************************************
nxosv1                     : ok=3    changed=3    unreachable=0    failed=0
```

3.3 テンプレート

このラボ演習では、Ansible プレイブックの出力結果を取得、保存、操作する方法を学びます。まず、プレイブックに変数を登録する方法と、この出力結果をディスク上のファイルに書き込む方法を確認します。

JSON の出力

最初の演習では、新しい簡単なプレイブックと Ansible NXOS モジュールを使用して、変数の登録や、変数の内容をファイルにコピーします。nxosv1 デバイスで show コマンドを実行し、利用可能な機能、有効な機能、無効な機能をリスト化します。

その次の演習では、nxos_command モジュールの詳細を確認します。この演習では、タスクの出力を登録する方法と、返されたコンテンツを「local_action」というタスクでコピーする方法に注目してください。

1. ansible ホストの Putty ターミナルで 3.3.templates ディレクトリに切り替えます。

```
[ansible@ansible ~]$ cd ~/dc-automation-bootcamp/nxos/lab3/3.3.templates
```

2. ansible の Putty ターミナルで次のコマンドを入力し、VI エディタで ansible プレイブックを表示/編集します。

```
[ansible@ansible 3.3.templates]$ vi interface_json.yml
```

プレイブック内のコメントは、各セクションの内容を説明しています。プレイブック全体を確認してその作用を理解してください。

注：使用する Ansible モジュール

nxos_command - http://docs.ansible.com/ansible/latest/nxos_command_module.html [英語]

local_action – タスクをターゲット デバイスで実行するのではなく、ansible プレイブックの起動元であるローカル マシンで実行するための、簡単なシンタックスです。local_action の詳細については、次のリンクを参照してください。
http://docs.ansible.com/ansible/latest/playbooks_delegation.html#delegation [英語]

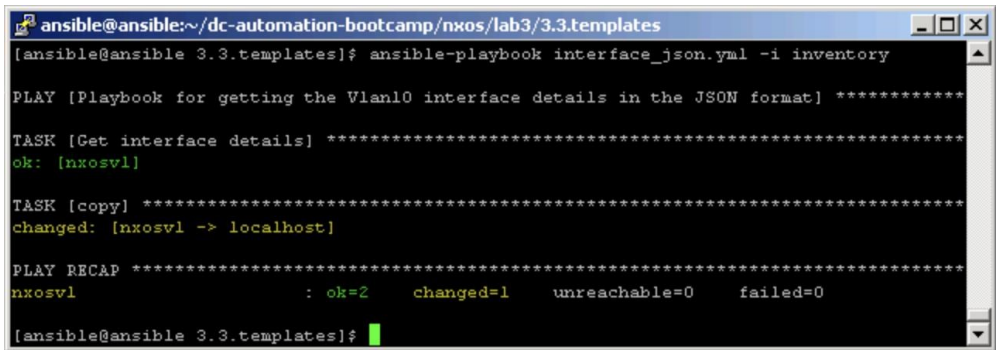
```
---
- name: Playbook for getting the Vlan10 interface details in the JSON format
  hosts: all
  gather_facts: no

  tasks:
  - name: Get interface details # Fetch Vlan10 interface details and store it in interface_facts
variable
    nxos_command:
      commands: "show ip interface Vlan10"
      transport: nxapi
    register: interface_facts

  # Copy the contents of the interface_facts variable into a json file
  - local_action: copy content={{ interface_facts }} dest=./intf-{{inventory_hostname}}.json
```

3. プレイブックをひととおり確認したら、VI エディタを終了します。

4. Putty ターミナルで以下のコマンドを入力して、ansible プレイブックを実行します。

```
[ansible@ansible 3.3.templates]$ ansible-playbook interface_json.yml -i inventory
```

実行後、端末には次のように表示されます。



出力結果

```
[ansible@ansible 3.3.templates]$ ansible-playbook interface_json.yml -i inventory

PLAY [Playbook for getting the Vlan10 interface details in the JSON format] ****************************

TASK [Get interface details] ***************************************************************************
ok: [nxosv1]

TASK [copy] ********************************************************************************************
changed: [nxosv1 -> localhost]

PLAY RECAP *********************************************************************************************
nxosv1                     : ok=2    changed=1    unreachable=0    failed=0
```

JSON の出力ファイル

次のコマンドを入力して、デバイスから取得されたデータを含む出力ファイルを表示します。

```
[ansible@ansible 3.3.templates]$ cat intf-nxosv1.json
```


```
{"failed": false, "changed": false, "stdout": [{"TABLE_vrf": {"ROW_vrf": {"vrf-name-out": "default"}},
"TABLE_intf": {"ROW_intf": {"upkt-orig": 0, "num-maddr": 0, "lbyte-recv": 0, "mpkt-consumed": 0, "proxy-
arp": "disabled", "stats-last-reset": "never", "upkt-recv": 0, "proto-state": "down", "upkt-consumed":
0, "bpkt-recv": 0, "prefix": "10.1.1.1", "tag": 0, "mpkt-fwd": 0, "num-addr": 1, "bbyte-consumed": 0,
"ip-disabled": "FALSE", "admin-state": "up", "upkt-fwd": 0, "subnet": "10.1.1.0", "lbyte-fwd": 0,
"bbyte-fwd": 0, "mbyte-sent": 0, "pref": 0, "bbyte-sent": 0, "bpkt-consumed": 0, "intf-name": "Vlan10",
"ubyte-fwd": 0, "lbyte-sent": 0, "upkt-sent": 0, "mpkt-recv": 0, "mbyte-orig": 0, "ubyte-consumed": 0,
"mrouting": "disabled", "bbyte-recv": 0, "port-unreach": "enabled", "ubyte-orig": 0, "link-state":
"down", "lcl-proxy-arp": "disabled", "mbyte-consumed": 0, "lpkt-sent": 0, "urpf-mode": "none", "bbyte-
orig": 0, "masklen": 24, "ip-ls-type": "none", "lbyte-consumed": 0, "ubyte-sent": 0, "bpkt-orig": 0,
"bpkt-fwd": 0, "mbyte-recv": 0, "iod": 72, "lpkt-consumed": 0, "lpkt-fwd": 0, "dir-bcast": "disabled",
"ubyte-recv": 0, "lpkt-orig": 0, "lpkt-recv": 0, "mpkt-orig": 0, "mpkt-sent": 0, "mtu": 1500, "bpkt-
sent": 0, "icmp-redirect": "enabled", "lbyte-orig": 0, "ip-unreach": 0, "mbyte-fwd": 0, "bcast-addr":
"255.255.255.255"}}}], "stdout_lines": [{"TABLE_vrf": {"ROW_vrf": {"vrf-name-out": "default"}},
"TABLE_intf": {"ROW_intf": {"upkt-orig": 0, "num-maddr": 0, "lbyte-recv": 0, "mpkt-consumed": 0, "proxy-
arp": "disabled", "stats-last-reset": "never", "upkt-recv": 0, "proto-state": "down", "upkt-consumed":
0, "bpkt-recv": 0, "prefix": "10.1.1.1", "tag": 0, "mpkt-fwd": 0, "num-addr": 1, "bbyte-consumed": 0,
"ip-disabled": "FALSE", "admin-state": "up", "upkt-fwd": 0, "subnet": "10.1.1.0", "lbyte-fwd": 0,
"bbyte-fwd": 0, "mbyte-sent": 0, "pref": 0, "bbyte-sent": 0, "bpkt-consumed": 0, "intf-name": "Vlan10",
"ubyte-fwd": 0, "lbyte-sent": 0, "upkt-sent": 0, "mpkt-recv": 0, "mbyte-orig": 0, "ubyte-consumed": 0,
"mrouting": "disabled", "bbyte-recv": 0, "port-unreach": "enabled", "ubyte-orig": 0, "link-state":
"down", "lcl-proxy-arp": "disabled", "mbyte-consumed": 0, "lpkt-sent": 0, "urpf-mode": "none", "bbyte-
orig": 0, "masklen": 24, "ip-ls-type": "none", "lbyte-consumed": 0, "ubyte-sent": 0, "bpkt-orig": 0,
"bpkt-fwd": 0, "mbyte-recv": 0, "iod": 72, "lpkt-consumed": 0, "lpkt-fwd": 0, "dir-bcast": "disabled",
```

```
"ubyte-recv": 0, "lpkt-orig": 0, "lpkt-recv": 0, "mpkt-orig": 0, "mpkt-sent": 0, "mtu": 1500, "bpkt-
sent": 0, "icmp-redirect": "enabled", "lbyte-orig": 0, "ip-unreach": 0, "mbyte-fwd": 0, "bcast-addr":
"255.255.255.255"}}}]}
```


注：上記出力結果の中で強調表示されている変数を、次のラボ演習およびタスクで使用します。

テンプレート ベースの出力

Ansible テンプレートを使用して、別の形式を利用したり、プレイブックの出力を変更したりすることができます。Ansible では、Jinja2 テンプレートを使用します。Jinja2 の詳細については、http://jinja.pocoo.org/docs/dev/ [英語] を参照してください。

1. 非常に基本的な Jinja2 テンプレートを使用して、前のプレイブックの出力結果を改善します。テンプレート ファイルを確認して、どのように定義されているか理解してください。次のコマンドを入力して、VI エディタでテンプレート ファイルを開きます。

```
[ansible@ansible 3.3.templates]$ vi intf_template.j2

Interface Details:

-----------------------------------------------
| Interface | Prefix | Subnet | Mask |
-----------------------------------------------
|   {{ interface }}   | {{ fmt_int.prefix }} | {{ fmt_int.subnet }} |   {{ fmt_int.masklen }}   |
-----------------------------------------------
```

2. ansible の Putty ターミナルで次のコマンドを入力し、VI エディタで ansible プレイブックを表示/編集します。

```
[ansible@ansible 3.3.templates]$ vi interface_template.yml
```

プレイブック内のコメントは、各セクションの内容を説明しています。プレイブック全体を確認してその作用を理解してください。

注：使用する Ansible モジュール

nxos_command - http://docs.ansible.com/ansible/latest/nxos_command_module.html [英語]

set_fact - http://docs.ansible.com/ansible/latest/set_fact_module.html [英語]

template - http://docs.ansible.com/ansible/latest/template_module.html [英語]

```
---
- name: Interface Configuration Playbook
  hosts: all
  gather_facts: no
  vars:
    interface: Vlan10

  tasks:
  - name: Get interface details # Fetch Vlan10 interface details and store it in interface_facts
variable
    nxos_command:
      commands: "show ip interface {{ interface }}"
      transport: nxapi
    register: interface_facts

  # Parse through the JSON and store only the interface related information in fmt_int
  variable - set_fact: fmt_int="{{ interface_facts.stdout[0].TABLE_intf.ROW_intf }}"
```

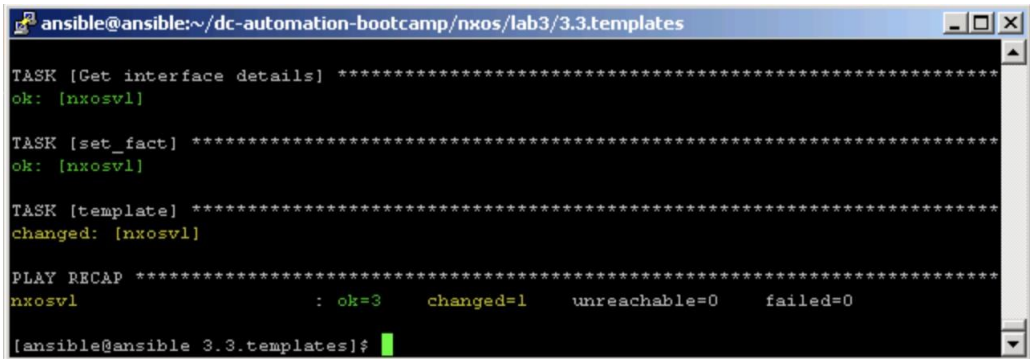
```
  # Create a new file using the intf_template.j2 template and substituting the variable values
  - template: src=./intf_template.j2 dest=./intf-{{inventory_hostname}}.txt
```

3. プレイブックをひととおり確認したら、VI エディタを終了します。

4. Putty ターミナルで以下のコマンドを入力して、ansible プレイブックを実行します。

```
[ansible@ansible 3.3.templates]$ ansible-playbook interface_template.yml -i inventory
```

実行後、端末には次のように表示されます。



出力結果

```
[ansible@ansible 3.3.templates]$ ansible-playbook interface_template.yml -i inventory

PLAY [Interface Configuration Playbook] ****************************************************

TASK [Get interface details] ***************************************************************
ok: [nxosv1]

TASK [set_fact] ****************************************************************************
ok: [nxosv1]

TASK [template] ****************************************************************************
ok: [nxosv1]

PLAY RECAP *********************************************************************************
nxosv1                     : ok=3    changed=0    unreachable=0    failed=0
```

出力ファイル

次のコマンドを入力して、デバイスから取得したデータを含む出力ファイルを表示します。テンプレートの使用方法に注目してください。出力ファイルがはるかに読みやすくなります。

```
[ansible@ansible 3.3.templates]$ cat intf-nxosv1.txt
Interface Details:

-----------------------------------------------
| Interface  |  Prefix   |  Subnet   |  Mask  |
-----------------------------------------------
|   Vlan10   | 10.1.1.1 | 10.1.1.0 |   24   |
-----------------------------------------------
```

これでシナリオ 1 のラボ 3 は完了です。

ラボ 4. Ansible の高度な用法

データ センター内では、多くのデバイスに共通する設定値がいくつかあります。そのため、適切な条件（機能、地理的ロケーションなど）でグループ化することで、自動化がより論理的に体系化され、効率が良くなります。最初のセクションでは、ホスト グループを定義することで効率的なプレイブックの記述に役立つことを確認していきます。2 番目のセクションでは、Ansible におけるロールの概念を理解します。ホスト グループのロールを定義すると、プレイブックをより論理的に体系化でき、その結果、拡張が容易になります。

4.1 グループ

1. ansible ホストの Putty ターミナルで 4.1.groups ディレクトリに切り替えます。

```
[ansible@ansible ~]$ cd ~/dc-automation-bootcamp/nxos/lab4/4.1.groups
```

2. インベントリ ファイルを確認し、ホストとグループがどのように定義されているか理解します。次のコマンドを入力して、VI エディタで外部変数ファイルを開きます。

```
[ansible@ansible 4.1.groups]$ vi inventory

nxosv1
nxosv2

[devtest]
nxosv1

[production]
nxosv2

[all:vars]
ansible_connection = local
ansible_ssh_user = ansible
ansible_ssh_pass = ansible
```

devtest と *production* という 2 つのグループが定義されました。それぞれのグループは 1 つのホストで構成されています。Ansible でホスト グループを定義すると、グループ レベルの変数（つまり、同一グループに属するすべてのデバイスに共通する設定変数）の定義に役立ちます。また、一連のホストに対してどのタスクを実行するかについても、グループを使用することで、より細かい制御が可能になります。

3. ansible の Putty ターミナルで次のコマンドを入力し、VI エディタで ansible プレイブックを表示/編集します。

```
[ansible@ansible 4.1.groups]$ vi main.yml
```

プレイブック内のコメントは、各セクションの内容を説明しています。プレイブック全体を確認してその作用を理解してください。

注：使用する Ansible モジュール

nxos_system - http://docs.ansible.com/ansible/latest/nxos_system_module.html [英語]

nxos_feature – http://docs.ansible.com/ansible/latest/nxos_feature_module.html [英語]

```
---
- name: Configuration Playbook using Groups
  hosts: all
  gather_facts: no
```

```
  # Ansible では、暗黙的に値がインポートされることはありません。決められたディレクトリ上にファイルが定義された際には自動でインポートされます。
  # もし、host_vars と group_vars で同じ値が利用されている場合、 host_vars で定義された値が利用されます。

  tasks:
  - name: configure hostname        # The variable is imported from the host vars directory
    nxos system:
      hostname: "{{ hostname }}"
      transport: nxapi

  - name: Enable features           # The variable is imported from the group vars directory
    nxos_feature:
      feature: "{{ item }}"
      state: enabled
      transport: nxapi
    with_items:
      "{{ features }}"
```

4. プレイブックをひととおり確認したら、VI エディタを終了します。

5. 次に、VI エディタを使用して、`group_vars` ディレクトリの下にあるグループ変数ファイルを確認します。

```
[ansible@ansible 4.1.groups]$ vi group_vars/devtest.yml

---

features:
 - interface-vlan
```

```
[ansible@ansible 4.1.groups]$ vi group_vars/production.yml

---

features:
 - interface-vlan
 - private-vlan
```

6. 今度は、VI エディタを使用して、`host_vars` ディレクトリの下にあるホスト変数ファイルを確認します。

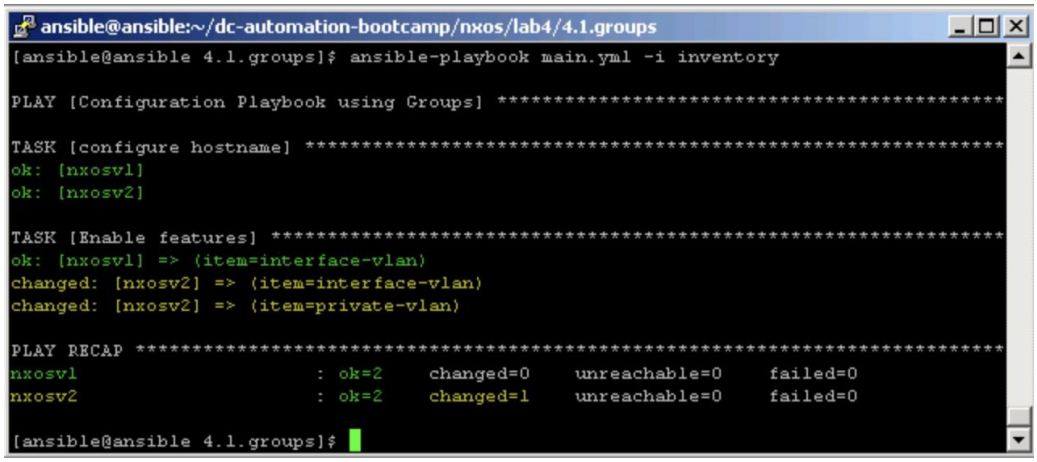
```
[ansible@ansible 4.1.groups]$ vi host_vars/nxosv1.yml

---

hostname: nxosv1
```

```
[ansible@ansible 4.1.groups]$ vi host_vars/nxosv2.yml

---

hostname: nxosv2
```

7. Putty ターミナルで以下のコマンドを入力して、ansible プレイブックを実行します。

```
[ansible@ansible 4.1.groups]$ ansible-playbook main.yml -i inventory
```

実行後、端末には次のように表示されます。



出力結果

```
[ansible@ansible 4.1.groups]$ ansible-playbook main.yml -i inventory

PLAY [Configuration Playbook using Groups] *************************************************************

TASK [configure hostname] ******************************************************************************
ok: [nxosv1]
ok: [nxosv2]

TASK [Enable features] *********************************************************************************
ok: [nxosv1] => (item=interface-vlan)
changed: [nxosv2] => (item=interface-vlan)
changed: [nxosv2] => (item=private-vlan)

PLAY RECAP *********************************************************************************************
nxosv1                     : ok=2    changed=0    unreachable=0    failed=0
nxosv2                     : ok=2    changed=1    unreachable=0    failed=0
```

4.2 ロール

ホストで実行される ansible タスクが大量にある場合、それを体系化するにはロールを使用するとよいでしょう。ホスト（今回のケースではスイッチ）には、ホストの機能と設定要件に基づいて、特定のロールを論理的に割り当てることができます。

ロールの最も重要な内容は、tasks と vars です。1 つのプレイブックに任意の数のロールをインポートできます。ロールをインポートすると、そのロールに対して定義されたすべてのタスクと変数が、自動的に読み込まれます。また、ロールごとにコンテンツをグループ化することによって、ロールを他のユーザと簡単に共有することもできます。

前の演習で使用したインベントリ ファイルには、production グループと devtest グループが設定されているため、そのまま使用します。

図 8. devtest デバイスの設定

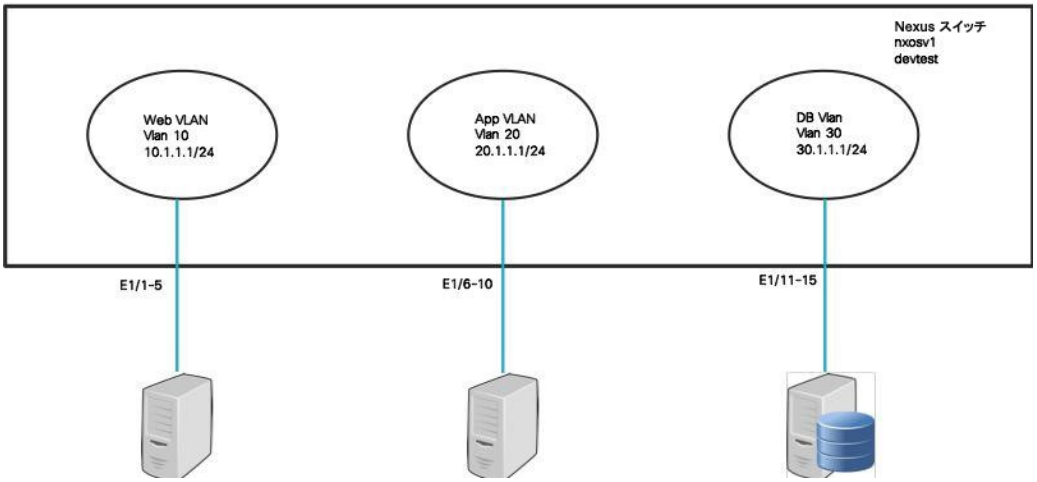
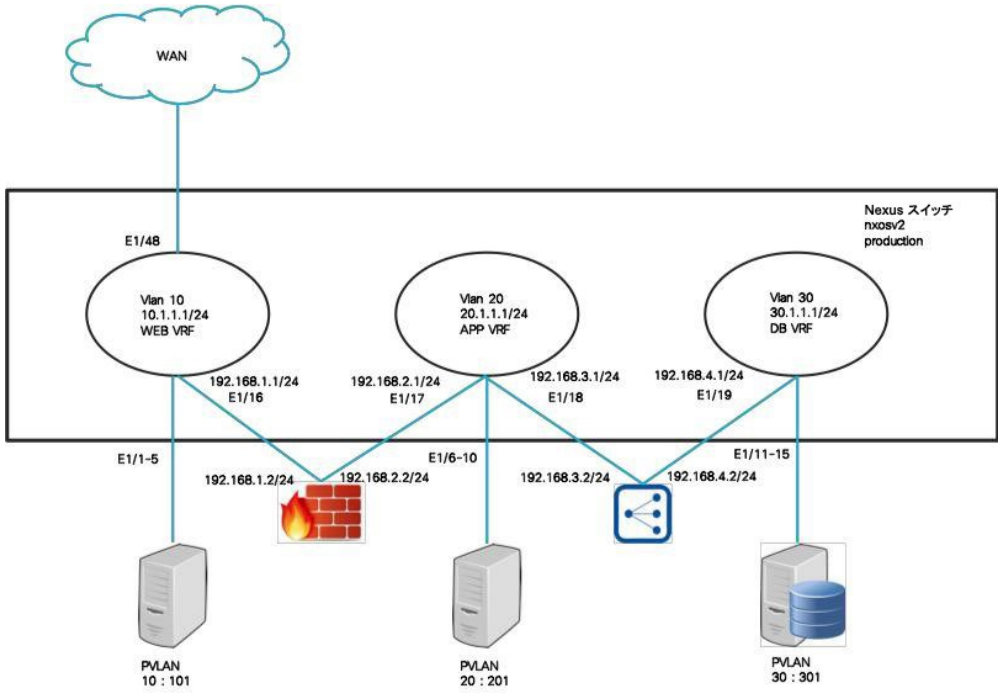


図 9. production デバイスの設定



1. ansible ホストの Putty ターミナルで 4.2.roles ディレクトリに切り替えます。

```
[ansible@ansible ~]$ cd ~/dc-automation-bootcamp/nxos/lab4/4.2.roles
```

2. *production*、*devtest*、*common* の 3 つのロールを作成します。作成したロールに必要なディレクトリとファイルの準備ができたら、ロールをプレイブックにインポートします。どのホスト グループにどのロールを割り当てるかについては、プレイブックで管理できます。devtest プレイブックを確認して、管理方法を理解してください。

```
[ansible@ansible 4.2.roles]$ vi devtest.yml

---
```

```
- name: Config playbook for devtest environment
  hosts: devtest# Run the playbook for only the devtest devices
  connection: local
  gather_facts: no
  roles:
  - devtest
```

3. common プレイブックには、本番環境とステージング環境の両方のホストに適用できる設定が含まれています。common プレイブックを開き、「:」（or 演算子）を使用することで devtest デバイスと production デバイスの両方がこのプレイブックの対象として含まれていることを確認します。

```
[ansible@ansible 4.2.roles]$ vi common.yml

---
- name: Config playbook for both production and devtest environment
  hosts: devtest:production # devtest と production の両方に対してプレイブックを実行する
  connection: local
  gather_facts: no
  roles:
  - common
```

4. 「roles」ディレクトリに各ロール用のディレクトリがあります。各ロールのディレクトリの中に、タスク、変数、ハンドラ用のディレクトリがあります。「devtest」ロールの tasks ディレクトリと vars ディレクトリの yaml ファイルを確認してみましょう。

```
[ansible@ansible 4.2.roles]$ vi roles/devtest/tasks/main.yml

---
- name: Set the switchport trunk allowed vlan on all physical
  interfaces nxos_switchport:
    interface: "{{ item.name }}"
    mode: trunk
    trunk_allowed_vlans: "1-4094"
    transport:
  nxapi with_items:
    "{{ physical_interfaces }}"

- name: Set the switchport access vlan
  nxos_switchport:
    interface: "{{ item.name }}"
    mode: access
    access_vlan: "{{ item.vlan }}"
    transport:
  nxapi with_items:
    "{{ physical_interfaces }}"

- name: IPv4 Address configuration for the VLAN interfaces
  nxos_ip_interface:
    interface: "{{ item.name }}"
    version: v4
    addr: "{{ item.ipv4 }}"
    mask: 24
    transport:
  nxapi with_items:
    "{{ vlan_interfaces }}"
```

```
[ansible@ansible 4.2.roles]$ vi roles/devtest/vars/main.yml
```

```
---

vlan_interfaces:
  - { name: Vlan10, ipv4: 10.1.1.1, mask: 24 }
  - { name: Vlan20, ipv4: 20.1.1.1, mask: 24 }
  - { name: Vlan30, ipv4: 30.1.1.1, mask: 24 }
```

注：各タスク内の main.yml ファイルを、さらに小さなプレイブックに分割することができます。production タスクの main.yml プレイブックは、同じディレクトリ内に存在する private_vlan.yml というもう 1 つのプレイブックをインポートします。private_vlan.yml プレイブックには、プライベート VLAN 設定に関連するすべてのタスクが含まれています。private_vlan 用の NXOS モジュールはまだ開発中で、公式の Ansible リリースでは利用できないため、ここでは汎用の ***nxos_config*** モジュールを使用して private_vlan を設定します。

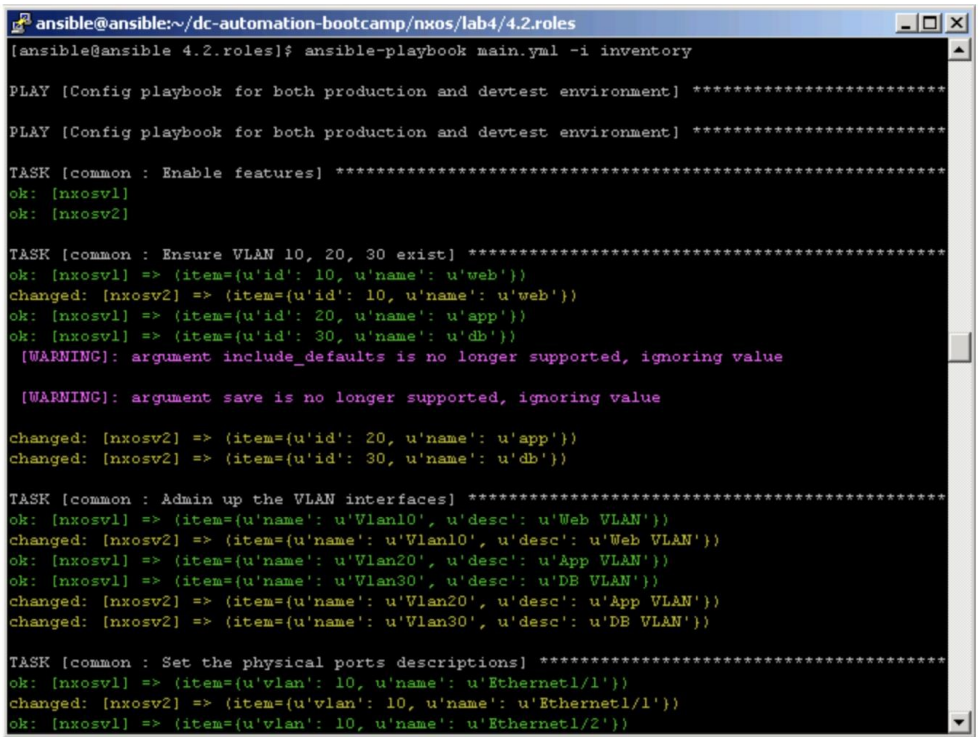
5. ansible の Putty ターミナルで次のコマンドを入力し、VI エディタで ansible の main プレイブックを表示/編集します。

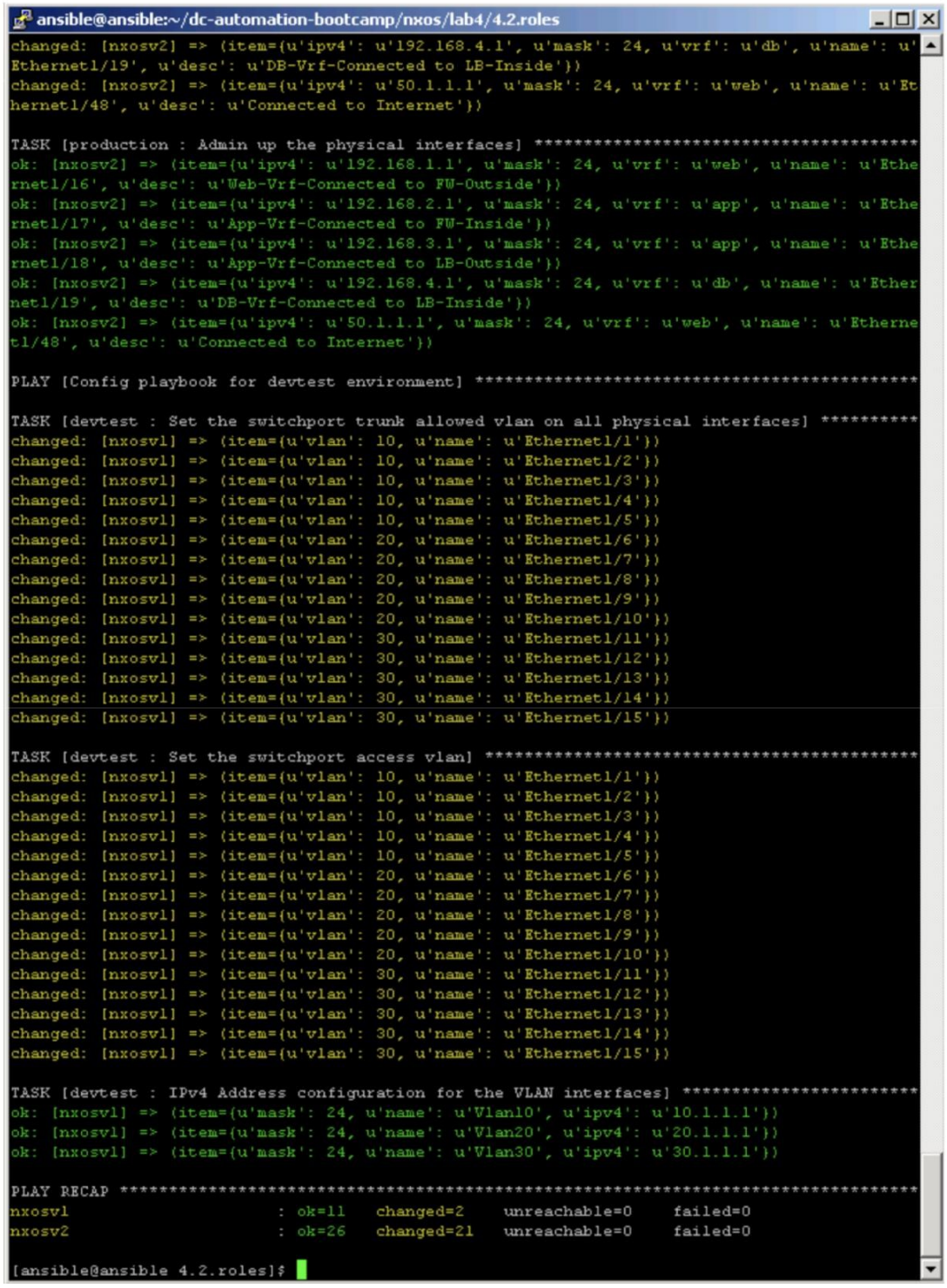
```
[ansible@ansible 4.2.roles]$ vi main.yml

---
- name: Config playbook for both production and devtest environment
  hosts: devtest:production
  connection: local
  gather_facts: no

- import_playbook: common.yml
- import_playbook: production.yml
- import_playbook: devtest.yml
```

6. Putty ターミナルで以下のコマンドを入力して、ansible プレイブックを実行します。

```
[ansible@ansible 4.2.roles]$ ansible-playbook main.yml -i inventory
```

実行後、端末には次のように表示されます。





出力結果

```
[ansible@ansible 4.2.roles]$ ansible-playbook main.yml -i inventory

PLAY [Config playbook for both production and devtest environment] *************************************
```

```
PLAY [Config playbook for both production and devtest environment] ************************************

TASK [common : Enable features] ***********************************************************************
ok: [nxosv2]
ok: [nxosv1]

TASK [common : Ensure VLAN 10, 20, 30 exist] **********************************************************
changed: [nxosv2] => (item={u'id': 10, u'name': u'web'})
ok: [nxosv1] => (item={u'id': 10, u'name': u'web'})
ok: [nxosv1] => (item={u'id': 20, u'name': u'app'})
ok: [nxosv1] => (item={u'id': 30, u'name': u'db'})
 [WARNING]: argument include_defaults is no longer supported, ignoring value

 [WARNING]: argument save is no longer supported, ignoring value

changed: [nxosv2] => (item={u'id': 20, u'name': u'app'})
changed: [nxosv2] => (item={u'id': 30, u'name': u'db'})

TASK [common : Admin up the VLAN interfaces] **********************************************************
ok: [nxosv1] => (item={u'name': u'Vlan10', u'desc': u'Web VLAN'})
ok: [nxosv1] => (item={u'name': u'Vlan20', u'desc': u'App VLAN'})
changed: [nxosv2] => (item={u'name': u'Vlan10', u'desc': u'Web VLAN'})
ok: [nxosv1] => (item={u'name': u'Vlan30', u'desc': u'DB VLAN'})
changed: [nxosv2] => (item={u'name': u'Vlan20', u'desc': u'App VLAN'})
changed: [nxosv2] => (item={u'name': u'Vlan30', u'desc': u'DB VLAN'})

TASK [common : Set the physical ports descriptions] ***************************************************
ok: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/1'})
changed: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/1'})
ok: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/2'})
ok: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/3'})
changed: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/2'})
ok: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/4'})
ok: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/5'})
skipping: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/6'})
skipping: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/7'})
skipping: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/8'})
skipping: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/9'})
skipping: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/10'})
skipping: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/11'})
skipping: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/12'})
skipping: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/13'})
skipping: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/14'})
skipping: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/15'})
changed: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/3'})
changed: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/4'})
changed: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/5'})
skipping: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/6'})
skipping: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/7'})
skipping: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/8'})
skipping: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/9'})
skipping: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/10'})
skipping: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/11'})
skipping: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/12'})
skipping: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/13'})
```

```
skipping: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/14'})
skipping: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/15'})

TASK [common : Set the physical ports descriptions] ****************************************************
skipping: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/1'})
skipping: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/2'})
skipping: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/1'})
skipping: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/2'})
skipping: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/3'})
skipping: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/3'})
skipping: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/4'})
skipping: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/4'})
skipping: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/5'})
skipping: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/5'})
ok: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/6'})
ok: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/7'})
changed: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/6'})
ok: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/8'})
changed: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/7'})
ok: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/9'})
ok: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/10'})
skipping: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/11'})
skipping: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/12'})
skipping: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/13'})
skipping: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/14'})
skipping: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/15'})
changed: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/8'})
changed: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/9'})
changed: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/10'})
skipping: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/11'})
skipping: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/12'})
skipping: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/13'})
skipping: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/14'})
skipping: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/15'})

TASK [common : Set the physical ports descriptions] ****************************************************
skipping: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/1'})
skipping: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/2'})
skipping: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/1'})
skipping: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/3'})
skipping: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/2'})
skipping: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/4'})
skipping: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/3'})
skipping: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/5'})
skipping: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/4'})
skipping: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/6'})
skipping: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/5'})
skipping: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/7'})
skipping: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/6'})
skipping: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/7'})
skipping: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/8'})
skipping: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/9'})
skipping: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/8'})
skipping: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/10'})
skipping: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/9'})
skipping: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/10'})
```

```
ok: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/11'})
ok: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/12'})
changed: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/11'})
ok: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/13'})
changed: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/12'})
ok: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/14'})
ok: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/15'})
changed: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/13'})
changed: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/14'})
changed: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/15'})

TASK [common : Set the physical interfaces eth1-15 as L2 ports] ****************************************
ok: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/1'})
ok: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/1'})
ok: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/2'})
ok: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/2'})
ok: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/3'})
ok: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/3'})
ok: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/4'})
ok: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/4'})
ok: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/5'})
ok: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/5'})
ok: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/6'})
ok: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/6'})
ok: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/7'})
ok: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/7'})
ok: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/8'})
ok: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/8'})
ok: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/9'})
ok: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/9'})
ok: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/10'})
ok: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/10'})
ok: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/11'})
ok: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/11'})
ok: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/12'})
ok: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/12'})
ok: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/13'})
ok: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/14'})
ok: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/13'})
ok: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/15'})
ok: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/14'})
ok: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/15'})

TASK [common : Admin up the physical interfaces] *******************************************************
ok: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/1'})
ok: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/1'})
ok: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/2'})
ok: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/2'})
ok: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/3'})
ok: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/3'})
ok: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/4'})
ok: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/4'})
ok: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/5'})
ok: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/5'})
ok: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/6'})
ok: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/6'})
```

```
ok: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/7'})
ok: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/7'})
ok: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/8'})
ok: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/8'})
ok: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/9'})
ok: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/9'})
ok: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/10'})
ok: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/10'})
ok: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/11'})
ok: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/11'})
ok: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/12'})
ok: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/12'})
ok: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/13'})
ok: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/13'})
ok: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/14'})
ok: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/14'})
ok: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/15'})
ok: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/15'})

PLAY [Config playbook for production environment] ******************************************************

TASK [production : Enable feature private-vlan] ********************************************************
ok: [nxosv2]

TASK [production : Create VRFs] ************************************************************************
changed: [nxosv2] => (item=App)
changed: [nxosv2] => (item=Web)
changed: [nxosv2] => (item=DB)

TASK [production : Create private vlans] ***************************************************************
changed: [nxosv2] => (item={u'vrf': u'Web', u'vlan': 10, u'mask': 24, u'ipv4': u'10.1.1.1',
u'private_vlan'    : 101})
changed: [nxosv2] => (item={u'vrf': u'App', u'vlan': 20, u'mask': 24, u'ipv4': u'20.1.1.1',
u'private vlan'    : 201})
changed: [nxosv2] => (item={u'vrf': u'DB', u'vlan': 30, u'mask': 24, u'ipv4': u'30.1.1.1',
u'private vlan':     301})

TASK [production : Associate private vlans with vlans] *************************************************
changed: [nxosv2] => (item={u'vrf': u'Web', u'vlan': 10, u'mask': 24, u'ipv4': u'10.1.1.1',
u'private vlan'    : 101})
changed: [nxosv2] => (item={u'vrf': u'App', u'vlan': 20, u'mask': 24, u'ipv4': u'20.1.1.1',
u'private vlan'    : 201})
changed: [nxosv2] => (item={u'vrf': u'DB', u'vlan': 30, u'mask': 24, u'ipv4': u'30.1.1.1',
u'private vlan':     301})

TASK [production : Private vlan mapping with Vlan interfaces] ******************************************
changed: [nxosv2] => (item={u'vrf': u'Web', u'vlan': 10, u'mask': 24, u'ipv4': u'10.1.1.1',
u'private_vlan'    : 101})
changed: [nxosv2] => (item={u'vrf': u'App', u'vlan': 20, u'mask': 24, u'ipv4': u'20.1.1.1',
u'private_vlan'    : 201})
changed: [nxosv2] => (item={u'vrf': u'DB', u'vlan': 30, u'mask': 24, u'ipv4': u'30.1.1.1',
u'private vlan':     301})

TASK [production : Associate private vlans with Physical interfaces] ***********************************
changed: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/1'})
changed: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/2'})
changed: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/3'})
changed: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/4'})
```

```
changed: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/5'})
skipping: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/6'})
skipping: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/7'})
skipping: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/8'})
skipping: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/9'})
skipping: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/10'})
skipping: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/11'})
skipping: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/12'})
skipping: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/13'})
skipping: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/14'})
skipping: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/15'})

TASK [production : Associate private vlans with Physical interfaces] ***********************************
skipping: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/1'})
skipping: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/2'})
skipping: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/3'})
skipping: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/4'})
skipping: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/5'})
changed: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/6'})
changed: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/7'})
changed: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/8'})
changed: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/9'})
changed: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/10'})
skipping: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/11'})
skipping: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/12'})
skipping: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/13'})
skipping: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/14'})
skipping: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/15'})

TASK [production : Associate private vlans with Physical interfaces] ***********************************
skipping: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/1'})
skipping: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/2'})
skipping: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/3'})
skipping: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/4'})
skipping: [nxosv2] => (item={u'vlan': 10, u'name': u'Ethernet1/5'})
skipping: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/6'})
skipping: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/7'})
skipping: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/8'})
skipping: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/9'})
skipping: [nxosv2] => (item={u'vlan': 20, u'name': u'Ethernet1/10'})
changed: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/11'})
changed: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/12'})
changed: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/13'})
changed: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/14'})
changed: [nxosv2] => (item={u'vlan': 30, u'name': u'Ethernet1/15'})

TASK [production : Create static routes associated with the VRFs] ***************************************
changed: [nxosv2] => (item={u'dest': u'0.0.0.0/0', u'next_hop': u'192.168.2.2', u'vrf': u'App'})
changed: [nxosv2] => (item={u'dest': u'30.1.1.0/24', u'next_hop': u'192.168.3.2', u'vrf': u'App'})
changed: [nxosv2] => (item={u'dest': u'0.0.0.0/0', u'next_hop': u'192.168.4.2', u'vrf': u'DB'})
changed: [nxosv2] => (item={u'dest': u'0.0.0.0/0', u'next_hop': u'50.1.1.2', u'vrf': u'Web'})
changed: [nxosv2] => (item={u'dest': u'20.1.1.0/24', u'next_hop': u'192.168.1.2', u'vrf': u'Web'})

TASK [production : Create IP Access List] ***************************************************************
changed: [nxosv2]
```

```
TASK [production : Apply ACL on  Web VLAN] ************************************************************
changed: [nxosv2]

TASK [production : Ensure respective VRFs exist on Vlan interfaces] ***********************************
changed: [nxosv2] => (item={u'vrf': u'Web', u'vlan': 10, u'mask': 24, u'ipv4': u'10.1.1.1',
u'private_vlan'   : 101})
changed: [nxosv2] => (item={u'vrf': u'App', u'vlan': 20, u'mask': 24, u'ipv4': u'20.1.1.1',
u'private_vlan'   : 201})
changed: [nxosv2] => (item={u'vrf': u'DB', u'vlan': 30, u'mask': 24, u'ipv4': u'30.1.1.1',
u'private_vlan':    301})

TASK [production : IPv4 Address configuration for the VLAN interfaces] ********************************
changed: [nxosv2] => (item={u'vrf': u'Web', u'vlan': 10, u'mask': 24, u'ipv4': u'10.1.1.1',
u'private_vlan'   : 101})
changed: [nxosv2] => (item={u'vrf': u'App', u'vlan': 20, u'mask': 24, u'ipv4': u'20.1.1.1',
u'private_vlan'   : 201})
changed: [nxosv2] => (item={u'vrf': u'DB', u'vlan': 30, u'mask': 24, u'ipv4': u'30.1.1.1',
u'private_vlan':    301})

TASK [production : Set the physical ports descriptions] ***********************************************
changed: [nxosv2] => (item={u'ipv4': u'192.168.1.1', u'mask': 24, u'vrf': u'web', u'name':
u'Ethernet1/16',    u'desc': u'Web-Vrf-Connected to FW-Outside'})
changed: [nxosv2] => (item={u'ipv4': u'192.168.2.1', u'mask': 24, u'vrf': u'app', u'name':
u'Ethernet1/17',    u'desc': u'App-Vrf-Connected to FW-Inside'})
changed: [nxosv2] => (item={u'ipv4': u'192.168.3.1', u'mask': 24, u'vrf': u'app', u'name':
u'Ethernet1/18',    u'desc': u'App-Vrf-Connected to LB-Outside'})
changed: [nxosv2] => (item={u'ipv4': u'192.168.4.1', u'mask': 24, u'vrf': u'db', u'name':
u'Ethernet1/19',    u'desc': u'DB-Vrf-Connected to LB-Inside'})
changed: [nxosv2] => (item={u'ipv4': u'50.1.1.1', u'mask': 24, u'vrf': u'web', u'name': u'Ethernet1/48',
u'   desc': u'Connected to Internet'})

TASK [production : Ensure interfaces are Layer 3 ports before configuring VRF] ************************
changed: [nxosv2] => (item={u'ipv4': u'192.168.1.1', u'mask': 24, u'vrf': u'web', u'name':
u'Ethernet1/16',    u'desc': u'Web-Vrf-Connected to FW-Outside'})
changed: [nxosv2] => (item={u'ipv4': u'192.168.2.1', u'mask': 24, u'vrf': u'app', u'name':
u'Ethernet1/17',    u'desc': u'App-Vrf-Connected to FW-Inside'})
changed: [nxosv2] => (item={u'ipv4': u'192.168.3.1', u'mask': 24, u'vrf': u'app', u'name':
u'Ethernet1/18',    u'desc': u'App-Vrf-Connected to LB-Outside'})
changed: [nxosv2] => (item={u'ipv4': u'192.168.4.1', u'mask': 24, u'vrf': u'db', u'name':
u'Ethernet1/19',    u'desc': u'DB-Vrf-Connected to LB-Inside'})
changed: [nxosv2] => (item={u'ipv4': u'50.1.1.1', u'mask': 24, u'vrf': u'web', u'name': u'Ethernet1/48',
u'   desc': u'Connected to Internet'})

TASK [production : Ensure respective vrf exists on Physical interfaces] *******************************
changed: [nxosv2] => (item={u'ipv4': u'192.168.1.1', u'mask': 24, u'vrf': u'web', u'name':
u'Ethernet1/16',    u'desc': u'Web-Vrf-Connected to FW-Outside'})
changed: [nxosv2] => (item={u'ipv4': u'192.168.2.1', u'mask': 24, u'vrf': u'app', u'name':
u'Ethernet1/17',    u'desc': u'App-Vrf-Connected to FW-Inside'})
changed: [nxosv2] => (item={u'ipv4': u'192.168.3.1', u'mask': 24, u'vrf': u'app', u'name':
u'Ethernet1/18',    u'desc': u'App-Vrf-Connected to LB-Outside'})
changed: [nxosv2] => (item={u'ipv4': u'192.168.4.1', u'mask': 24, u'vrf': u'db', u'name':
u'Ethernet1/19',    u'desc': u'DB-Vrf-Connected to LB-Inside'})
changed: [nxosv2] => (item={u'ipv4': u'50.1.1.1', u'mask': 24, u'vrf': u'web', u'name': u'Ethernet1/48',
u'   desc': u'Connected to Internet'})
 [WARNING]: The VRF is not present/active on the device. Use nxos_vrf to fix this.


TASK [production : IPv4 Address configuration for the physical interfaces]*********************
changed: [nxosv2] => (item={u'ipv4': u'192.168.1.1', u'mask': 24, u'vrf': u'web', u'name':
u'Ethernet1/16',    u'desc': u'Web-Vrf-Connected to FW-Outside'})
```

```
changed: [nxosv2] => (item={u'ipv4': u'192.168.2.1', u'mask': 24, u'vrf': u'app', u'name':
u'Ethernet1/17',    u'desc': u'App-Vrf-Connected to FW-Inside'})
changed: [nxosv2] => (item={u'ipv4': u'192.168.3.1', u'mask': 24, u'vrf': u'app', u'name':
u'Ethernet1/18',    u'desc': u'App-Vrf-Connected to LB-Outside'})
changed: [nxosv2] => (item={u'ipv4': u'192.168.4.1', u'mask': 24, u'vrf': u'db', u'name':
u'Ethernet1/19',    u'desc': u'DB-Vrf-Connected to LB-Inside'})
changed: [nxosv2] => (item={u'ipv4': u'50.1.1.1', u'mask': 24, u'vrf': u'web', u'name': u'Ethernet1/48',
u'   desc': u'Connected to Internet'})

TASK [production : Admin up the physical interfaces] ***************************************************
ok: [nxosv2] => (item={u'ipv4': u'192.168.1.1', u'mask': 24, u'vrf': u'web', u'name': u'Ethernet1/16',
u'de   sc': u'Web-Vrf-Connected to FW-Outside'})
ok: [nxosv2] => (item={u'ipv4': u'192.168.2.1', u'mask': 24, u'vrf': u'app', u'name': u'Ethernet1/17',
u'de   sc': u'App-Vrf-Connected to FW-Inside'})
ok: [nxosv2] => (item={u'ipv4': u'192.168.3.1', u'mask': 24, u'vrf': u'app', u'name': u'Ethernet1/18',
u'de   sc': u'App-Vrf-Connected to LB-Outside'})
ok: [nxosv2] => (item={u'ipv4': u'192.168.4.1', u'mask': 24, u'vrf': u'db', u'name': u'Ethernet1/19',
u'des   c': u'DB-Vrf-Connected to LB-Inside'})
ok: [nxosv2] => (item={u'ipv4': u'50.1.1.1', u'mask': 24, u'vrf': u'web', u'name': u'Ethernet1/48',
u'desc'   : u'Connected to Internet'})

PLAY [Config playbook for devtest environment] *********************************************************

TASK [devtest : Set the switchport trunk allowed vlan on all physical interfaces] **********************
changed: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/1'})
changed: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/2'})
changed: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/3'})
changed: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/4'})
changed: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/5'})
changed: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/6'})
changed: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/7'})
changed: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/8'})
changed: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/9'})
changed: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/10'})
changed: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/11'})
changed: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/12'})
changed: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/13'})
changed: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/14'})
changed: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/15'})

TASK [devtest : Set the switchport access vlan] ********************************************************
changed: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/1'})
changed: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/2'})
changed: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/3'})
changed: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/4'})
changed: [nxosv1] => (item={u'vlan': 10, u'name': u'Ethernet1/5'})
changed: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/6'})
changed: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/7'})
changed: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/8'})
changed: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/9'})
changed: [nxosv1] => (item={u'vlan': 20, u'name': u'Ethernet1/10'})
changed: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/11'})
changed: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/12'})
changed: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/13'})
changed: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/14'})
changed: [nxosv1] => (item={u'vlan': 30, u'name': u'Ethernet1/15'})

TASK [devtest : IPv4 Address configuration for the VLAN interfaces] ************************************
```

```
ok: [nxosv1] => (item={u'mask': 24, u'name': u'Vlan10', u'ipv4': u'10.1.1.1'})
ok: [nxosv1] => (item={u'mask': 24, u'name': u'Vlan20', u'ipv4': u'20.1.1.1'})
ok: [nxosv1] => (item={u'mask': 24, u'name': u'Vlan30', u'ipv4': u'30.1.1.1'})

PLAY RECAP *********************************************************************************************
nxosv1                     : ok=11   changed=2    unreachable=0    failed=0
nxosv2                     : ok=26   changed=21   unreachable=0    failed=0
```

これでシナリオ 1 のラボ 4 は完了です。

シナリオ 2. ACI の自動化

価値提案：ACI には、自動化に使用できる数種類のインターフェイスがあります。このシナリオでは、ACI の REST API および Ansible を使用してさまざまな設定タスクを自動化する方法を学習します。4 つのラボ演習を通し、Ansible のさまざまな用法を利用して一般的な設定タスクを実施します。

ACI には、自動化に使用できる数種類のインターフェイスがあります。

REST API: HTTP

ACI ツールキット：Python モジュール（簡易版）

Cobra SDK：Python モジュール（完全版）

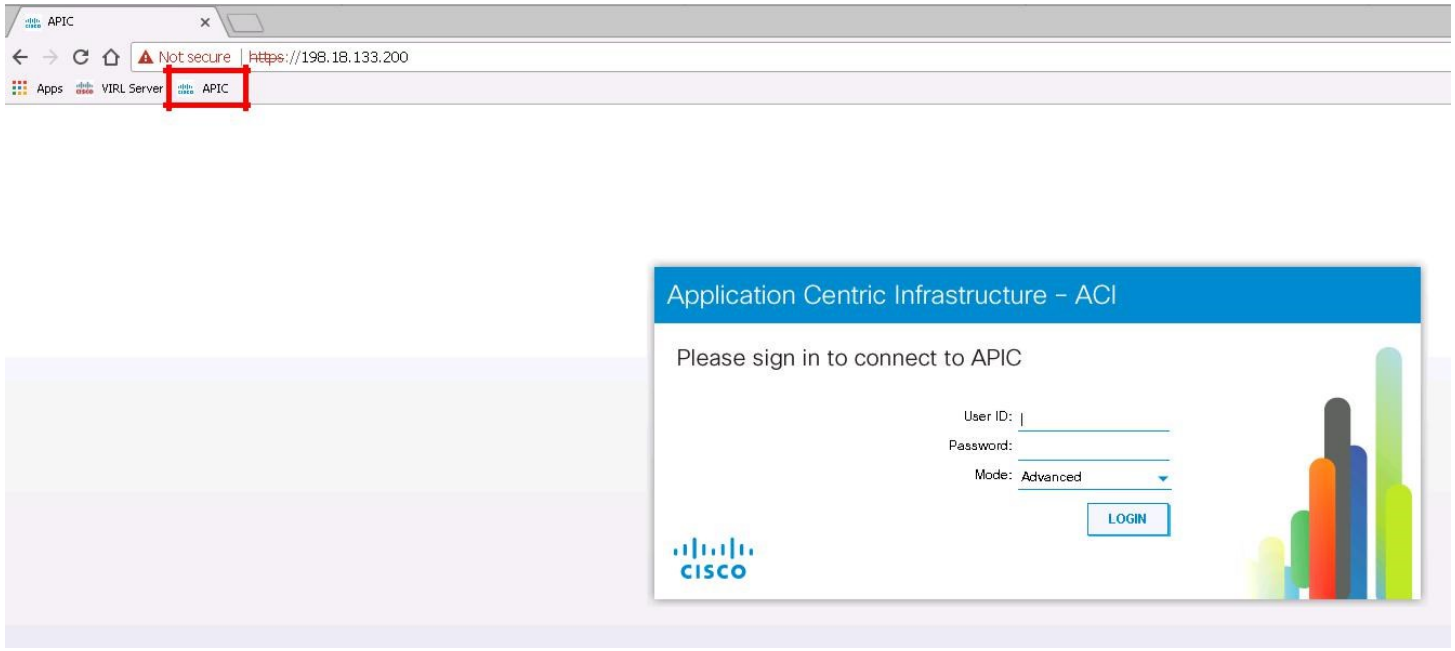
Arya：XML または JSON を Python コードに変換するツール

Ansible：Cisco Ansible モジュール

以降のラボ演習では、まず、Python スクリプトを利用した ACI 自動化について確認し、その後、Ansible による自動化について確認します。

1. dCloud ラボには、これらのラボ演習を実施するために使用できる APIC シミュレータが含まれています。ACI の GUI は、リモート ワークステーションのブラウザでアクセスできます。APIC シミュレータの GUI は、*https://198.18.133.200*（chrome にブックマークされています）からアクセスできます。ユーザ名は ***admin***、パスワードは **C1sco12345** です。

図 10. APIC ログイン画面



Python

このセクションでは、Python スクリプトを使用した ACI の自動化を紹介します。Python の自動化スクリプトや REST API を使用して ACI に CLI コマンドを挿入する方法を確認します。

ラボ 1：Python の基本的コンセプト

CLI の自動化

2. リモート ワークステーションのデスクトップで、「ansible」ホストに接続する Putty ショートカットを起動します。

3. このラボ演習では、Python を使用して ACI に CLI コマンドを挿入します。この方法を使って ACILab_clitest というテナントを作成します。ansible の Putty ターミナルで次のコマンドを入力し、VI エディタで Python スクリプトを表示/編集します。

```
[ansible@ansible ~]$ vi ~/dc-automation-bootcamp/aci/python/lab1/create_tenant_cli.py
```

スクリプト内のコメントは、各コマンドの内容を説明しています。スクリプト全体を確認してその作用を理解してください。

```
import paramiko
import time

# a list of the hosts we wish to access
hosts = ["apic"]

# NXOS login details
username = "admin"
password = "C1sco12345"

# Create a new Paramiko SSH connection
object conn = paramiko.SSHClient()
# Automatically add SSH hosts keys
conn.set_missing_host_key_policy(paramiko.AutoAddPolicy())

# 接続対象のホスト全てに対して実行する
for host in hosts:
        print "--------------------",host,"--------------------"
        # 複数コマンドを実行するために shell のセッションを作成する
        conn.connect(host, 22, username, password, look_for_keys=False, allow_agent=False)
        remote_shell = conn.invoke_shell()
        # リモート ホストのシェルの出力を受信
        # output = remote_shell.recv(65535)
        # display the output
        print output

        # send the command "configure terminal"
        remote_shell.send("configure terminal\n")
        time.sleep(.5)
        output = remote_shell.recv(65535)
```

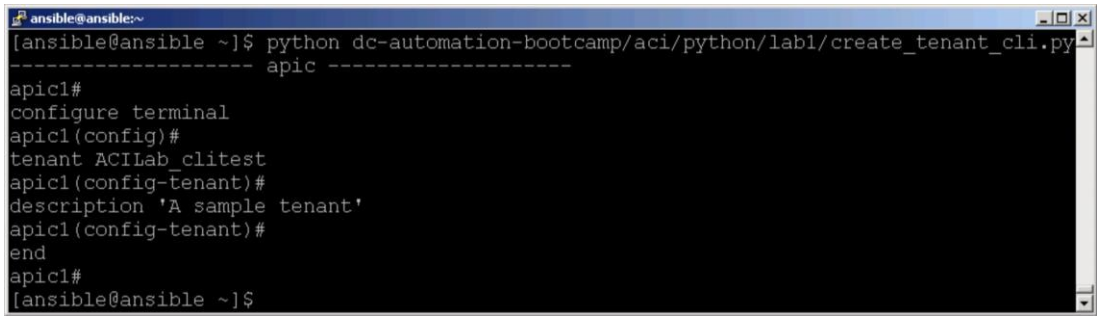
```
        print output

        # Create a tenant
        remote_shell.send("tenant ACILab_clitest\n")
        time.sleep(.5)
        output = remote_shell.recv(65535)
        print output

        # set description for the tenant
        remote_shell.send("description 'A sample tenant'\n")
        time.sleep(.5)
        output = remote_shell.recv(65535)
        print output

             # exit the configuration
        mode remote_shell.send("end\n")
        time.sleep(.5)
        output = remote_shell.recv(65535)
        print output
        time.sleep(.5)

        # close the SSH session to this host we can re-use the object for the
        # next host
        conn.close()
```

4. スクリプトをひととおり確認したら、VI エディタを終了します。

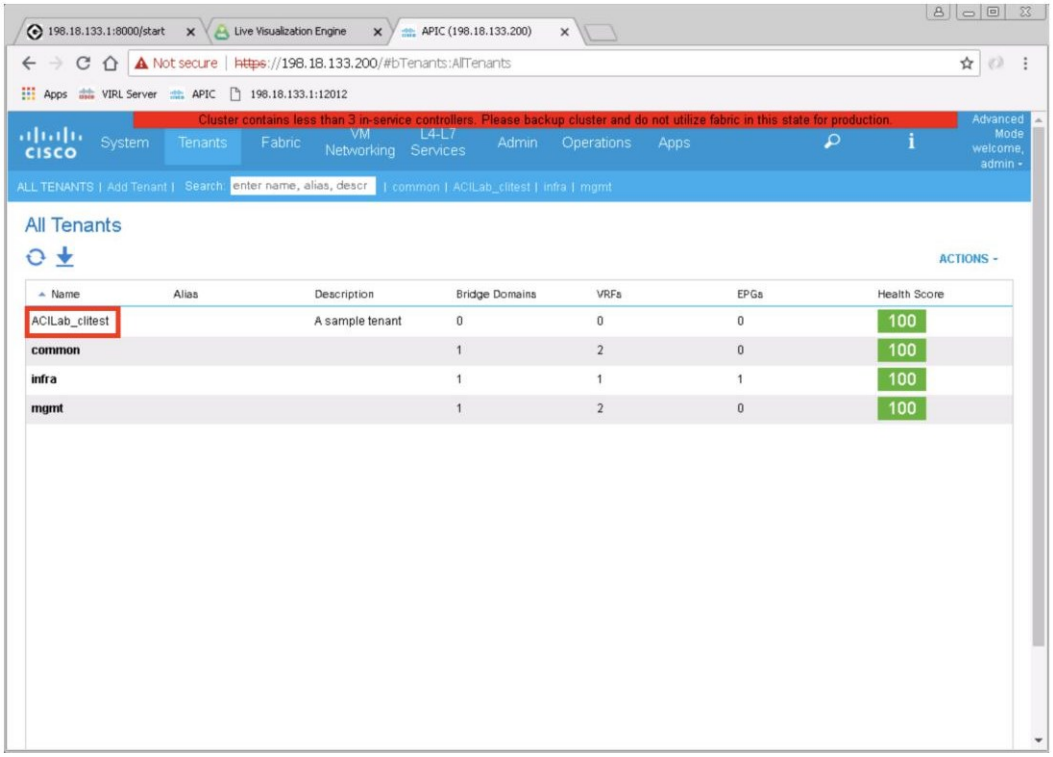
5. Putty ターミナルで以下のコマンドを入力して、ラボ演習を実行します。

```
[ansible@ansible ~]$ python ~/dc-automation-bootcamp/aci/python/lab1/create_tenant_cli.py
```

実行後、端末には次のように表示されます。



6. ここで、リモート デスクトップ システムの Web ブラウザを使用して、APIC コントローラの Web UI にアクセスできます。[テナント（Tenants）] タブに移動し、前のスクリプトの実行結果を確認します。



この方法では、簡単に自動化できますが、信頼性はあまり高くありません。たとえば上記のスクリプトでは、待機時間が短いとスクリプトが失敗する上、そのようなエラーの検出が容易ではありません。

REST API の自動化

ACI タスクを自動化するには、REST API を使用するほうが適しています。REST API は、*Postman* や *Curl* など、任意の HTTP クライアントから実行できます。自動化するには、Python の requests ライブラリを使用して REST API を実行します。REST API を使用してテナントを作成するためのスクリプトを、以下に示します。

1. ansible の Putty ターミナルで次のコマンドを入力し、VI エディタで Python スクリプトを表示/編集します。

```
[ansible@ansible ~]$ vi ~/dc-automation-bootcamp/aci/python/lab1/create_tenant_rest.py
```

スクリプト内のコメントは、各コマンドの内容を説明しています。スクリプト全体を確認してその作用を理解してください。

```
import requests
import urllib3

urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

# a list of the hosts we wish to access
hosts = ["apic"]

# For each host we wish to connect to
for host in hosts:
    print "\nAuthenticating with the device . . .\n"
    # Get an authentication token for the device
    url = "https://" + host + "/api/aaaLogin.json"
    # Login data
    data = """
```

```
    {
      "aaaUser": {
        "attributes": {
          "name": "admin",
          "pwd": "C1sco12345"
        }
      }
    }
    """
     response = requests.post(url, data=data, headers={'Content-Type': 'application/json'},
verify=False)
    # print json.dumps(response.json(), indent=2)

    if response.status_code == requests.codes.ok:
        print "Authentication successful!\n"
    else:
        print "Authentication failed! Please verify login credentials!\n"
        exit(0)

    token = response.json()['imdata'][0]['aaaLogin']['attributes']['token']
    print "Authentication Token: " + token + "\n"

    # Create tenant
    url = "https://" + host + "/api/mo/uni.json"
    # Tenant configuration
    data data = """
        {
            "fvTenant":
                { "attributes":
                {
                    "name": "ACILab_apitest",
                    "descr": "A test tenant"
                }
            }
        }
    """
    cookie = {'APIC-Cookie': token}
    response = requests.post(url, cookies=cookie, data=data, headers={'Content-Type':
'application/json'}, verify=False)

    if response.status_code == requests.codes.ok:
        print "Tenant created successfully!\n"
    else:
        print "Tenant creation failed!\n"
        exit(0)
```

2. スクリプトをひととおり確認したら、VI エディタを終了します。

3. Putty ターミナルで以下のコマンドを入力して、ラボ演習を実行します。

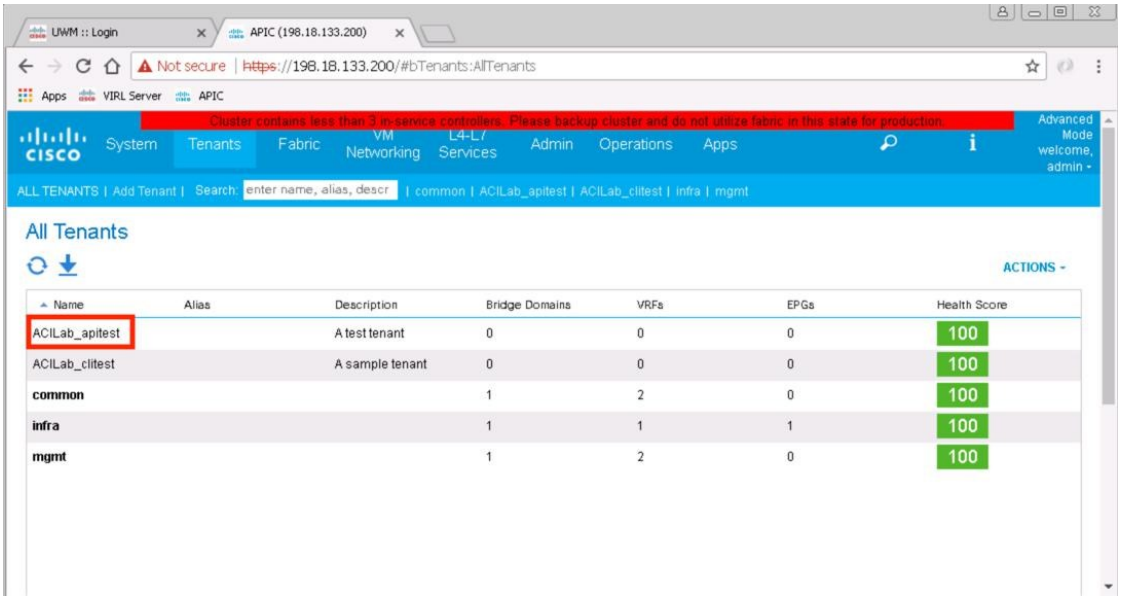
```
[ansible@ansible ~]$ python ~/dc-automation-bootcamp/aci/python/lab1/create_tenant_rest.py
```

4. 実行後、端末には次のように表示されます。

```
ansible@ansible:~
[ansible@ansible ~]$ python dc-automation-bootcamp/aci/python/lab1/create_tenant_rest.p
y

Authenticating with the device . . .

Authentication successful!

Authentication Token: Up69M45itSNo7DAYa783BLbsog58Icsi22mxzKllTZF5tUbl4CPUqrMj1x+96Jpx7
DjbOjQrPTQ9Y6N9zuklv4cKXcQLBZCCMU62Tc/C3JdWWAaNq9mLmO02kqJEtzeA5v4atyrlCr+qU6fD35FoFUjv
3OFZWhT4KhidO7XWxCstkY7ex66QnSbFGQoQ4/Rm

Tenant created successfully!

[ansible@ansible ~]$
```

5. ここで、リモート デスクトップ システムの Web ブラウザを使用して、APIC コントローラの Web UI にアクセスできます。[テナント（Tenants）] タブに移動し、前のスクリプトの実行結果を確認します。

図 11. 新しく作成されたテナント



これでシナリオ 2 のラボ 1 は完了です。

ラボ 2：Python でのコンフィギュレーション ファイルの使用

前のラボ演習で使用した自動化スクリプトでは、設定値がスクリプトに直接書かれていました。同じスクリプトを別のデバイスや設定値で使用する場合、スクリプト自体を変更する必要が生じるので、望ましいやり方とは言えません。設定値を変更する際に、誤ってスクリプトも変更してしまう可能性があるからです。これを防ぐために、スクリプトから値を取り出して、別のファイルに保存することができます。

このラボ演習では、VRF を作成します。VRF の設定データは、外部ファイルに保存されます。

CLI の自動化

1. ansible ホストの Putty ターミナルで lab2 ディレクトリに切り替えます。

```
[ansible@ansible ~]$ cd ~/dc-automation-bootcamp/aci/python/lab2
```

2. 設定値は、自動化スクリプト自体ではなく、外部ファイルに保存されています。次のコマンドを入力して、外部変数ファイルを確認してみましょう。

```
[ansible@ansible lab2]$ vi python_cli_host_data
```

```
apic,22,admin,C1sco12345,ACILab_clitest,ACILab_VRF,A sample VRF
```

3. 次に、以下のコマンドを入力して、VI エディタで Python スクリプトを開きます。

```
[ansible@ansible lab2]$ vi create_vrf_cli.py
```

前のラボの自動化 CLI スクリプトと比較して、コードに新たな行が追加されていることを確認します。これは、変数ファイルから設定値を取得するためのコードです。

```
#!/usr/bin/env python2

import paramiko
import time

# create an empty host dictionary
# we store the hosts in here and then SSH to each one in turn with the
# associated values used for configuration
host_dict = {}

# first, read in the external host data fields
with open("python_cli_host_data") as f:
    for line in f:
        split_line = line.split(',')
        key = split_line[0]
        val  =  split_line[1:]
        host_dict[key] = val

# now we have a dictionary of hosts and configuration values, we can proceed to
# SSH to the hosts and make the configuration changes

# Create a new Paramiko SSH connection object
conn = paramiko.SSHClient()
# Automatically add SSH hosts keys
conn.set_missing_host_key_policy(paramiko.AutoAddPolicy())
```

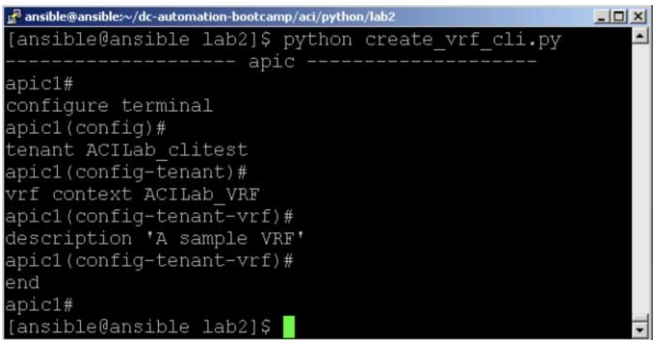
```
for host in host_dict.keys():
    (port,username, password, tenant, vrf_name, vrf_description) = host_dict[host]

    print "-------------------- ",host,"--------------------- "
    # create a shell session for multiple commands
    conn.connect(host, int(port), username, password, look_for_keys=False, allow_agent=False)
    remote_shell = conn.invoke_shell()
    # receive remote host shell output
    output = remote_shell.recv(65535)
    # display the output
    print output

    # send the command "configure terminal"
    remote_shell.send("configure terminal\n")
    time.sleep(.5)
    output = remote_shell.recv(65535)
    print output

    # Select tenant
    remote_shell.send("tenant "+ tenant+"\n")
    time.sleep(.5)
    output = remote_shell.recv(65535)
    print output

    # Create VRF
    remote_shell.send("vrf context " + vrf_name +"\n")
    time.sleep(.5)
    output = remote_shell.recv(65535)
    print output

    # set description for VRF
    remote_shell.send("description '" + vrf_description + "'\n")
    time.sleep(.5)
    output = remote_shell.recv(65535)
    print output
    # exit the configuration mode
    remote_shell.send("end\n")
    time.sleep(.5)
    output = remote_shell.recv(65535)
    print output
    time.sleep(.5)

    # close the SSH session to this host we can re-use the object for the
    # next host
    conn.close()
```

4. スクリプトをひととおり確認したら、VI エディタを終了します。

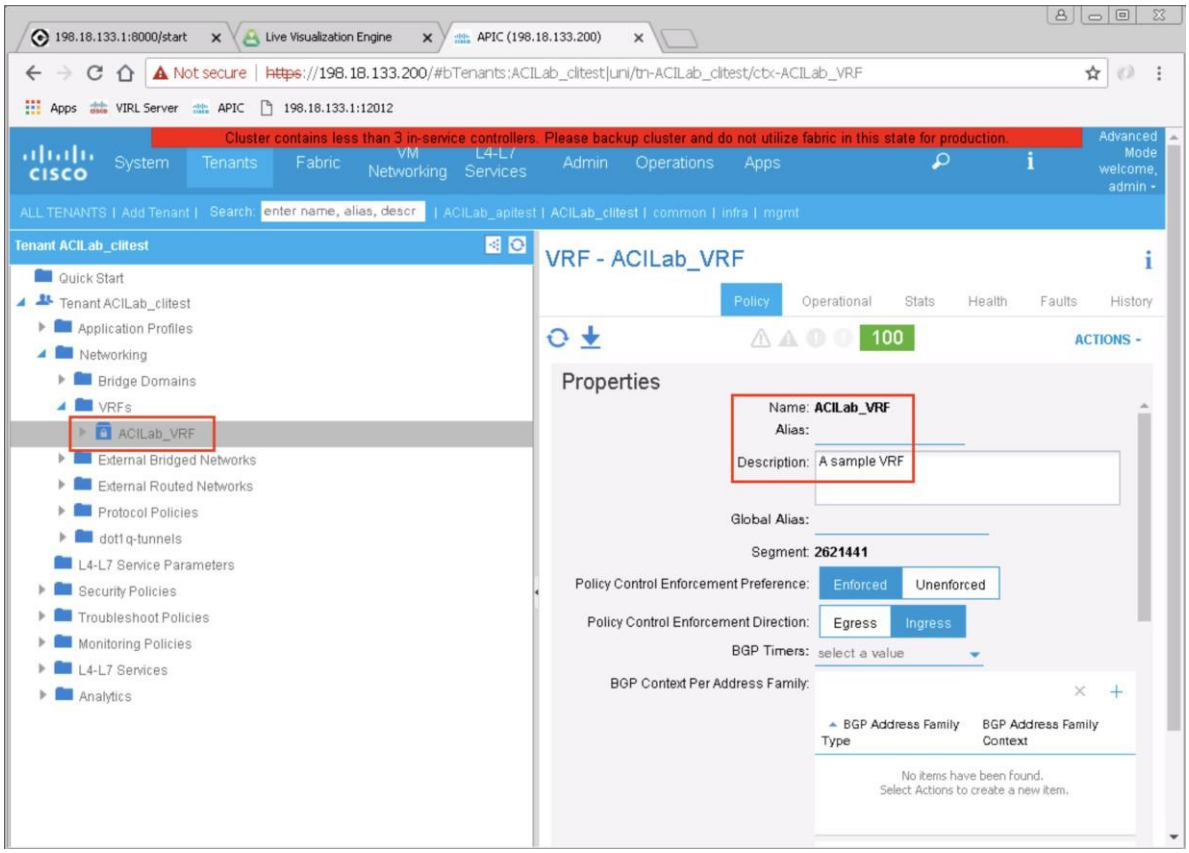
5. Putty ターミナルで以下のコマンドを入力して、ラボ演習を実行します。

```
[ansible@ansible lab2]$ python create_vrf_cli.py
```

6. 実行後、端末には次のように表示されます。



7. ここで、リモート デスクトップ システムの Web ブラウザを使用して、APIC コントローラの Web UI にアクセスします。[テナント（Tenants）] タブに移動し、「ACILab_clitest」テナントをダブルクリックします。テナントのウィンドウが開いたら、[ネットワー キング（Networking）] > [VRFs] の順にフォルダを展開し、新しく作成した [ACILab_VRF] を表示します。ACILab_VRF へのフル パスは、**ACILab_clitest/Networking/VRFs/AVILab_VRF** です。



REST API の自動化

1. 設定値は、自動化スクリプト自体ではなく、外部ファイルに保存されています。次のコマンドを入力して、外部変数ファイルを確認してみましょう。

```
[ansible@ansible lab2]$ vi python_rest_host_data
apic,22,admin,C1sco12345,ACILab_apitest,ACILab_VRF,A sample VRF
```

2. 次に、以下のコマンドを入力して、VI エディタで Python スクリプトを開きます。

```
[ansible@ansible lab2]$ vi create_vrf_rest.py
```

前のラボの自動化 CLI スクリプトと比較して、コードに新たな行が追加されていることを確認します。これは、変数ファイルから設定値を取得するためのコードです。

```
import requests
import urllib3
from ansible.plugins.callback import json

urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

# a list of the hosts we wish to access
# create an empty host dictionary
# we store the hosts in here and then SSH to each one in turn with the
# associated values used for configuration
host_dict = {}

# first, read in the external host data fields
with open("python_rest_host_data") as f:
    for line in f:
        split_line = line.split(',')
        key = split_line[0]
        val  =  split_line[1:]
        host_dict[key] = val

# For each host we wish to connect to
for host in host_dict.keys():
    (port, username, password, tenant, vrf_name, vrf_description) = host_dict[host]
    print "\nAuthenticating with the device . . .\n"
    # Get an authentication token for the device url
       = "https://" + host + "/api/aaaLogin.json"
    # Login data
    data = """
    {
      "aaaUser": {
        "attributes": {
          "name": "%s",
          "pwd": "%s"
        }
      }
    }
    """ % (username, password)
    response = requests.post(url, data=data, headers={'Content-Type': 'application/json'}, verify=False)
    # print json.dumps(response.json(), indent=2)

    if response.status_code == requests.codes.ok:
        print "Authentication successful!\n"
    else:
        print "Authentication failed! Please verify login credentials!\n"
        exit(0)

    token = response.json()['imdata'][0]['aaaLogin']['attributes']['token']
    print "Authentication Token: " + token + "\n"
```

```
    # Create VRF
    url = "https://" + host + "/api/mo/uni/tn-%s.json" % tenant
    # VRF configuration
    data data = """
        {
          "fvCtx":
              { "attributes":
              {
                "name": "%s",
                "descr": "%s"
          }
        }

        """ % (vrf_name, vrf_description.strip('\n'))

cookie = {'APIC-Cookie': token}

    response = requests.post(url, cookies=cookie, data=data, headers={'Content-Type':
'application/json'}, verify=False)
```

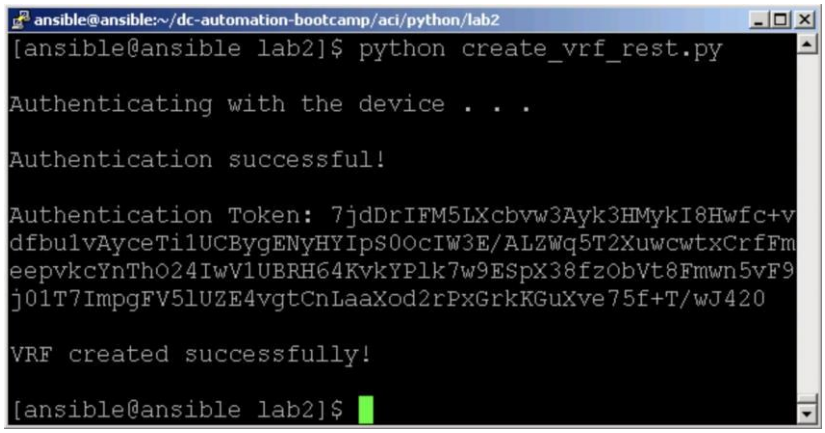
```
    if response.status_code == requests.codes.ok:
        print "VRF created successfully!\n"
    else:
        print "VRF creation failed!\n"
        exit(0)
```

3. スクリプトをひととおり確認したら、VI エディタを終了します。

4. Putty ターミナルで以下のコマンドを入力して、ラボ演習を実行します。

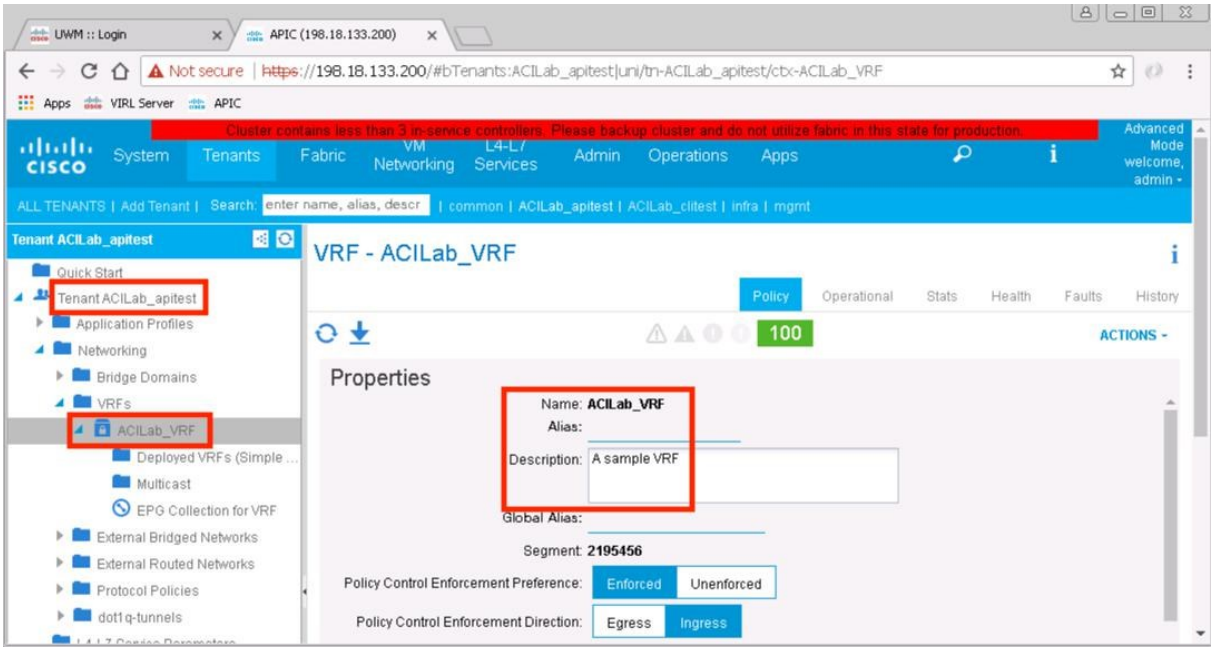
```
[ansible@ansible lab2]$ python create_vrf_rest.py
```

5. 実行後、端末には次のように表示されます。



6. ここで、リモート デスクトップ システムの Web ブラウザを使用して、APIC コントローラの Web UI にアクセスできます。[テナント（Tenants）] タブに移動し、 *「ACILab_apitest」* テナントをダブルクリックします。ページの左側で [ネットワーキング（Networking）] > [VRFs] フォルダの順に開きます。[VRFs] フォルダをダブルクリックすると、設定された VRF が表示されます。

図 12. 新しく作成された VRF



これでシナリオ 2 のラボ 2 は完了です。

Ansible

Ansible コア モジュールには、ACI 向けのものが 30 以上あります。このセクションでは、いくつかの ACI モジュールを使って、Ansible を使用した ACI 設定自動化の方法を学びます。各ラボ演習で作成/設定したオブジェクトを、その次の演習で利用します。すべてのラボ演習を記載どおりの順序で実行することを推奨します。

ラボ 3：テナントの作成

このラボ演習では、シンプルなプレイブックを使用してテナントを作成します。

1. ansible ホストの Putty ターミナルで aci/ansible/lab1 ディレクトリに切り替えます。

```
[ansible@ansible ~]$ cd ~/dc-automation-bootcamp/aci/ansible/lab1
```

2. ansible の Putty ターミナルで次のコマンドを入力し、VI エディタで ansible プレイブックを表示/編集します。

```
[ansible@ansible lab1]$ vi create_tenant.yml
```

プレイブック全体を確認してその作用を理解してください。Ansible で使用するシンタックスについては、NXOS の演習を通して、もうすでに使い慣れたことと思います。

注：使用する Ansible モジュール

aci_tenant - https://docs.ansible.com/ansible/latest/modules/aci_tenant_module.html **[英語]**

```
---
- name: playbook for testing tenants
  hosts: apic
  connection: local
  gather_facts: no

  tasks:

  - name: Add a new
    tenant aci_tenant:
      hostname: apic
      username: admin
      password: C1sco12345
      tenant: ACILab
      description: Test tenant
      state: present
      validate_certs: false
```

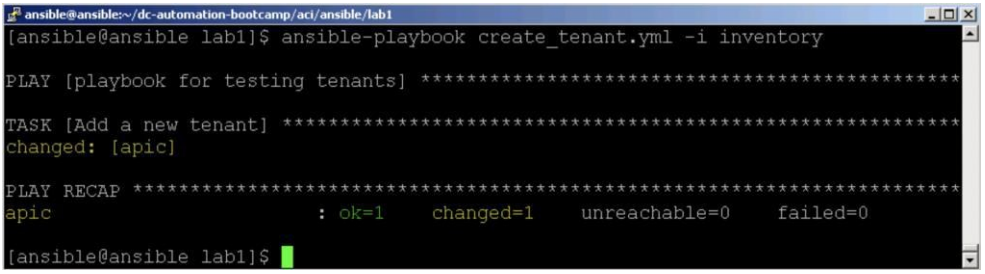
注：*validate_certs: false* を使用して、ansible が SSL 証明書を検証しないように設定しています。この設定は、自己署名証明書を使用して個人的に管理しているサイト以外では使用しないでください。

3. プレイブックをひととおり確認したら、VI エディタを終了します。

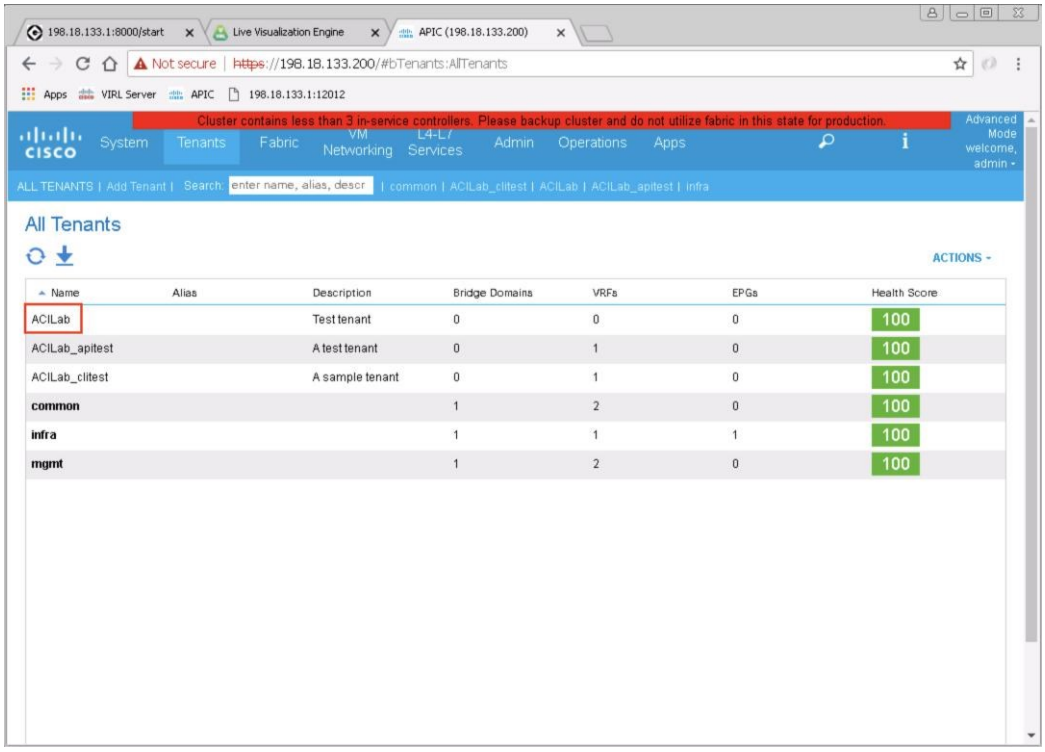
4. Putty ターミナルで以下のコマンドを入力して、ansible プレイブックを実行します。

```
[ansible@ansible lab1]$ ansible-playbook create_tenant.yml -i inventory
```

実行後、端末には次のように表示されます。



5. ここで、リモート デスクトップ システムの Web ブラウザを使用して、APIC コントローラの Web UI にアクセスできます。[テナント（Tenants）] タブに移動し、前のスクリプトの実行結果を確認します。



これでシナリオ 2 のラボ 3 は完了です。

ラボ 4：VRF およびブリッジ ドメインの作成

前の演習では、設定値は Ansible プレイブック内に保存されていました。さまざまな理由から、これは望ましい方法とは言えません。設定値を変更する必要がある場合に、誤ってプレイブックのコードも変更してしまうリスクが常に伴うからです。自動化のコードと設定値は分離しておくことがベスト プラクティスです。このラボ演習では、その方法を確認します。

設定値は、インベントリ ファイルまたは外部の変数ファイルに保管できます。インベントリ ファイルに保存されている値は Ansible によって自動的にインポートされますが、外部変数ファイルは明示的にインポートする必要があるので、注意してください。

1. ansible ホストの Putty ターミナルで aci/ansible/lab2 ディレクトリに切り替えます。

```
[ansible@ansible ~]$ cd ~/dc-automation-bootcamp/aci/ansible/lab2
```

2. いくつかの設定値が、インベントリ ファイル内の変数として保管されています。これらの値は、Ansible によって自動的にインポートされます。次のコマンドを入力して、インベントリ ファイルを確認します。

```
[ansible@ansible lab2]$ vi inventory
```

```
apic user=admin pass=C1sco12345
```

3. いくつかの設定値が、外部変数ファイル内の変数として保管されています。これらの値は、Ansible で明示的にインポートする必要があります。次のコマンドを入力して、外部変数ファイルを確認します。

```
[ansible@ansible lab2]$ vi external_vars.yml
```

```
---
vrf:
  name: ACILab_VRF
  description: A sample VRF used for ACILab
  tenant: ACILab

bd:
  name: ACILab_BD1
  vrf: ACILab_VRF
  tenant: ACILab
```

4. ansible の Putty ターミナルで次のコマンドを入力し、VI エディタで ansible プレイブックを表示/編集します。

```
[ansible@ansible lab2]$ vi create_vrf_bd.yml
```

プレイブック全体を確認してその作用を理解してください。プレイブックで外部変数ファイルを明示的にインポートする方法を確認してください。

注：使用する Ansible モジュール

aci_vrf - https://docs.ansible.com/ansible/2.5/modules/aci_vrf_module.html [英語]

aci_bd - https://docs.ansible.com/ansible/2.5/modules/aci_bd_module.html [英語]

```
---

- name: playbook for creating bd
  hosts: apic
```

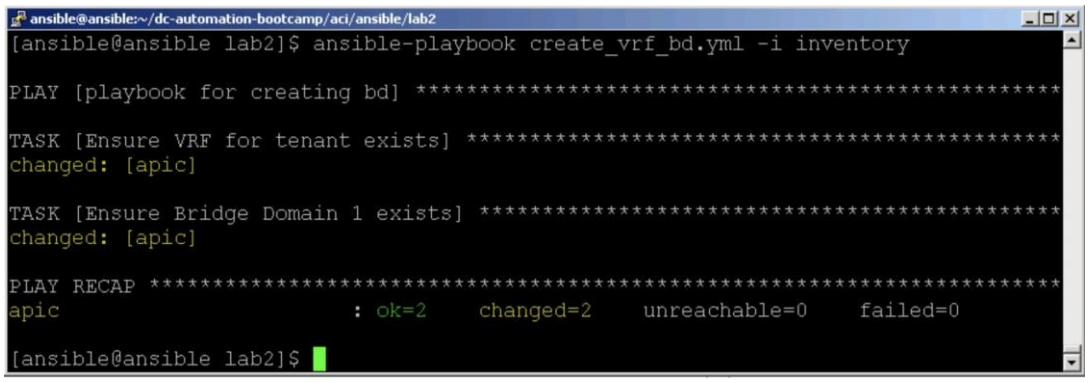
```
  connection: local
  gather_facts: no
  vars_files:
    - external_vars.yml

  tasks:
  - name: Ensure VRF for tenant
    exists aci_vrf:
      vrf: "{{ vrf.name }}"
      description: "{{ vrf.description }}"
      tenant: "{{ vrf.tenant }}"
      state: present
      host: "{{ inventory_hostname }}"
      username: "{{ user }}"
      password: "{{ pass }}"
      validate_certs: false

  - name: Ensure Bridge Domain 1
    exists aci_bd:
      bd: "{{ bd.name }}"
      vrf: "{{ bd.vrf }}"
      tenant: "{{ bd.tenant }}"
      state: present
      host: "{{ inventory_hostname }}"
      username: "{{ user }}"
      password: "{{ pass }}"
      validate_certs: false
```

5. プレイブックをひととおり確認したら、VI エディタを終了します。

6. Putty ターミナルで以下のコマンドを入力して、ansible プレイブックを実行します。

```
[ansible@ansible lab2]$ ansible-playbook create_vrf_bd.yml -i inventory
```

実行後、端末には次のように表示されます。



7. リモート デスクトップ システムの Web ブラウザを使用して、APIC コントローラの Web UI にアクセスします。[テナント（Tenants）] タブに移動し、「*ACILab*」テナントをダブルクリックします。ページの左側で [ネットワーキング（Networking）] フォルダを開き、BD と VRF が設定されていることを確認します。

図 13. 新しく作成されたブリッジ ドメイン

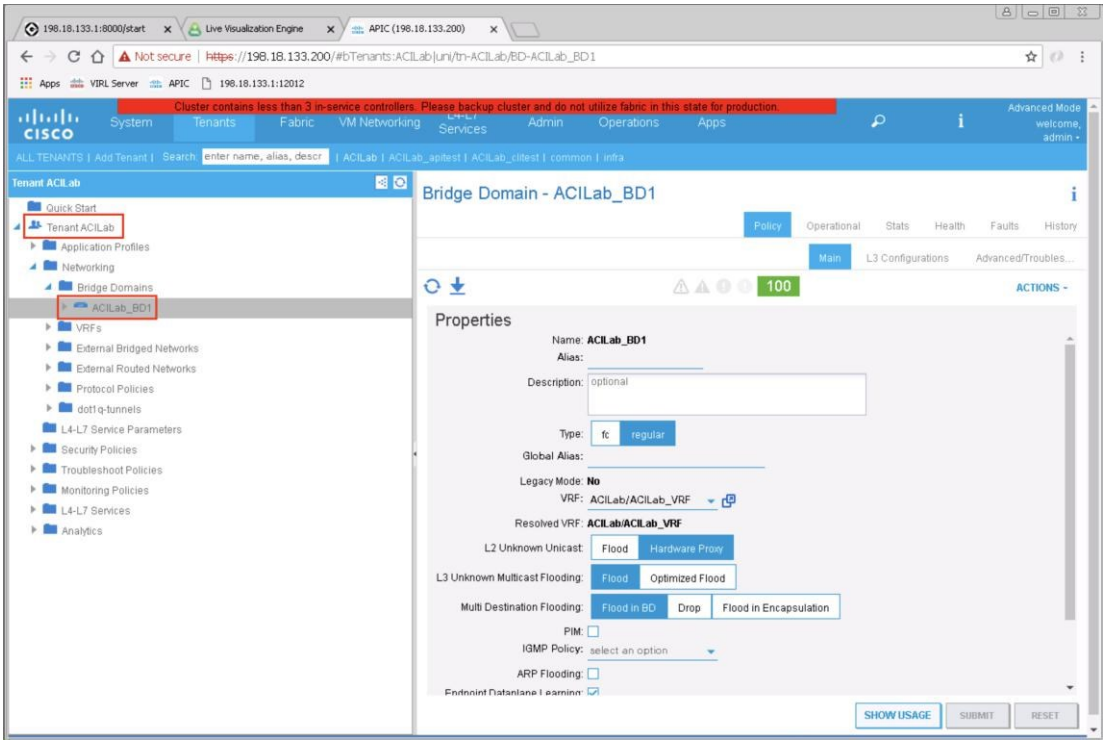
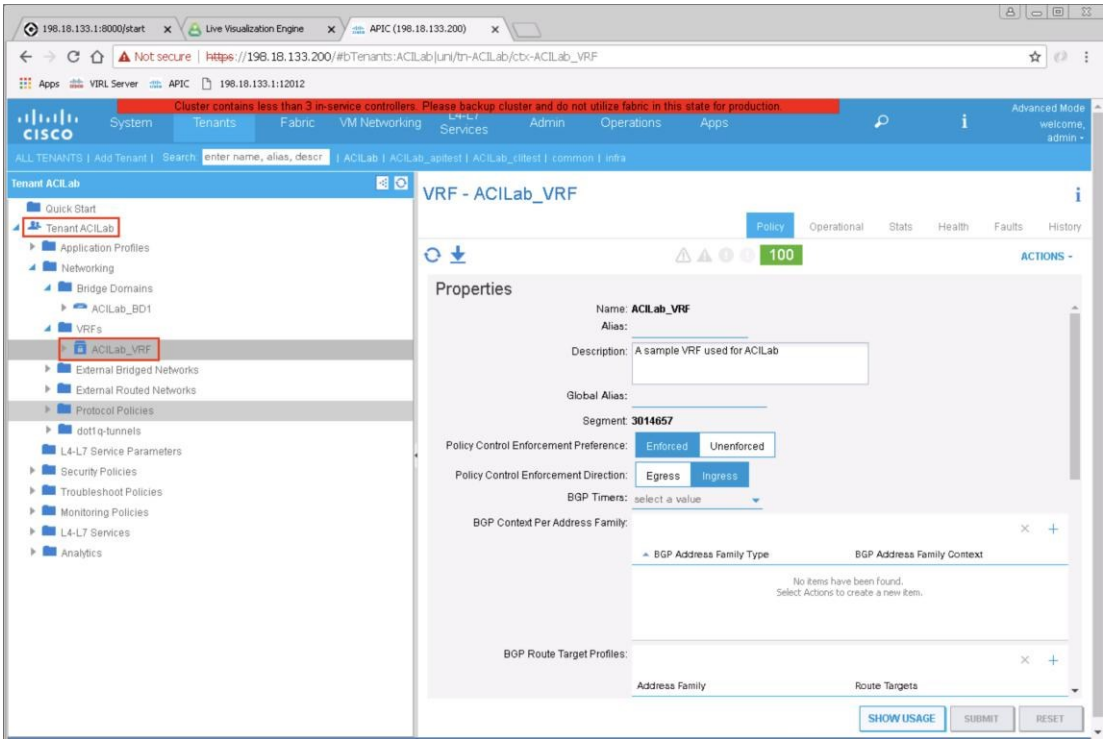


図 14. 新しく作成された VRF



これでシナリオ 2 のラボ 4 は完了です。

ラボ 5：コントラクトの作成

このラボ演習では、ループを使用して、同じコードの繰り返しを減らす方法を確認します。たとえば、複数のコントラクトを作成する必要がある場合に、それぞれのコントラクトごとにプレイブック タスクを記述する代わりに *with_items* ループを使用することができます。「with_items」を使用して反復処理するアイテムのタイプは、必ずしも文字列のリストである必要はありません。ハッシュのリスト（キーと値のペアおよびその他のデータ構造）を反復処理することも可能です。

このラボ演習では、プレイブックを使用して、web、app、db の 3 つのコントラクトを作成します。

1. ansible ホストの putty ターミナルで aci/ansible/lab3 ディレクトリに切り替えます。

```
[ansible@ansible ~]$ cd ~/dc-automation-bootcamp/aci/ansible/lab3
```

2. ansible の Putty ターミナルで次のコマンドを入力し、VI エディタで ansible プレイブックを表示/編集します。

```
[ansible@ansible lab3]$ vi create_contracts.yml
```

プレイブック全体を確認してその作用を理解してください。ansible タスク内でコントラクト名のリストがどのように定義されているか確認してください。

注：使用する Ansible モジュール

aci_contract - https://docs.ansible.com/ansible/2.5/modules/aci_contract_module.html [英語]

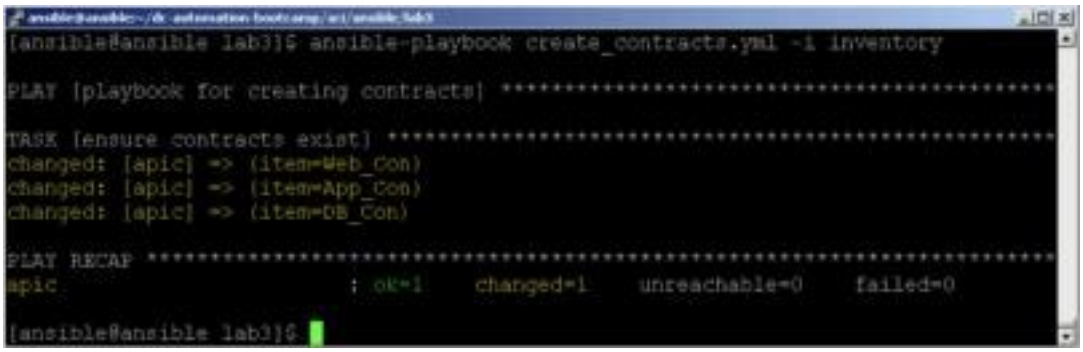
```
---

- name: playbook for creating contracts
  hosts: apic
  connection: local
  gather_facts: no

  tasks:

    - name: ensure contracts exist
      aci_contract:
        name: "{{ item }}"
        tenant: ACILab
        host: "{{ inventory_hostname }}"
        username: "{{ user }}"
        password: "{{ pass }}"
        validate_certs: false
      with_items:
        - Web_Con
        - App_Con
        - DB_Con
```

3. プレイブックをひととおり確認したら、VI エディタを終了します。

4. Putty ターミナルで以下のコマンドを入力して、ansible プレイブックを実行します。

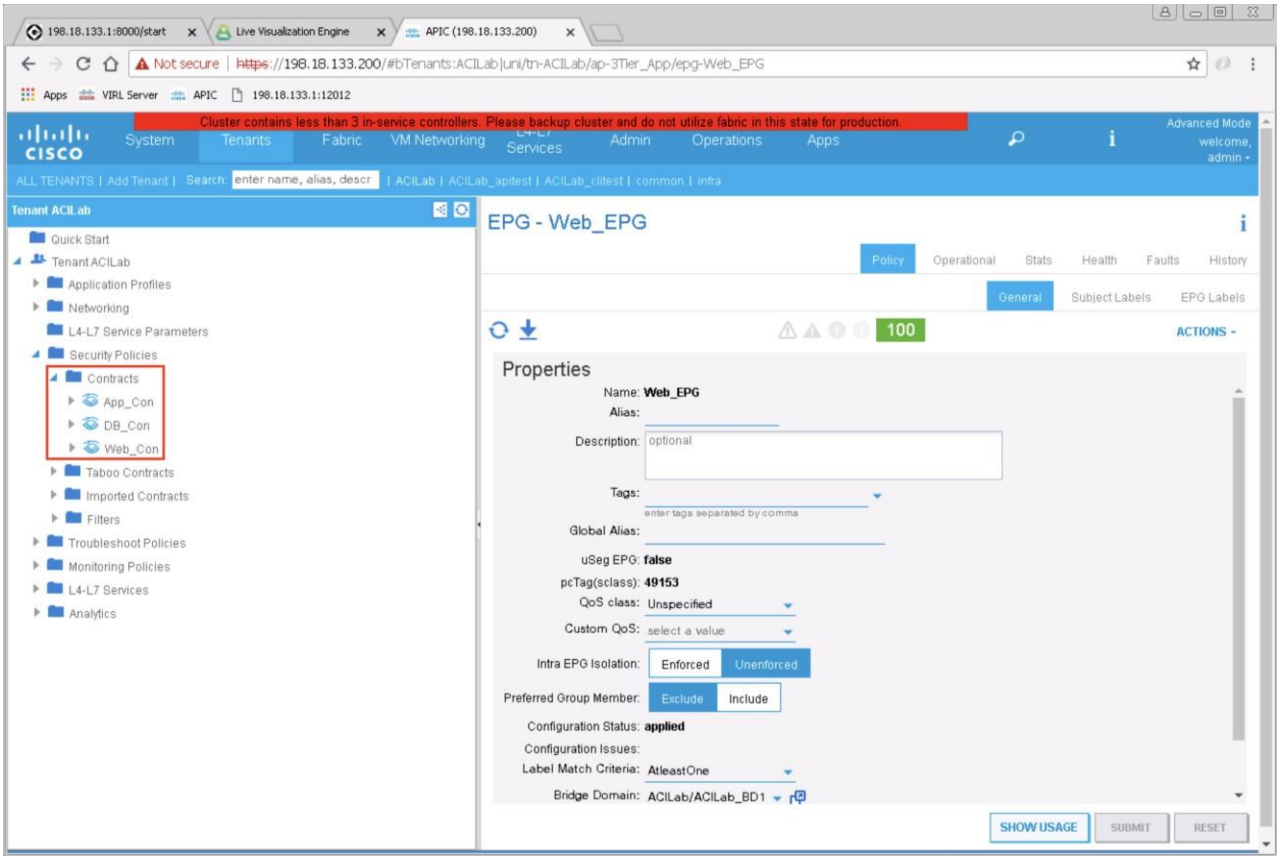
```
[ansible@ansible lab3]$ ansible-playbook create_contracts.yml -i inventory
```

実行後、端末には次のように表示されます。



5. ここで、リモート デスクトップ システムの Web ブラウザを使用して、APIC コントローラの Web UI にアクセスします。[テナント（Tenants）] タブに移動し、「ACILab」テナントをダブルクリックします。ページの左側で [セキュリティポリシー（Security Policies）] > [コントラクト（Contracts）] の順にフォルダを開き、コントラクトが設定されていることを確認します。

図 15. 新しく作成されたコントラクト



これでシナリオ 2 のラボ 5 は完了です。

ラボ 6：ANP と EPG の作成

このラボ演習では、エンド ポイント グループ（EPG）とアプリケーション ネットワーク プロファイル（ANP）を作成します。前のラボ演習で作成したテナントとブリッジ ドメインを使用します。

このラボ演習では、プレイブックを使用して、web、app、db の 3 つのコントラクトを作成します。

1. ansible ホストの Putty ターミナルで aci/ansible/lab4 ディレクトリに切り替えます。

```
[ansible@ansible ~]$ cd ~/dc-automation-bootcamp/aci/ansible/lab4
```

2. 設定値はプレイブック自体ではなく、外部ファイルに保存されています。次のコマンドを入力して、外部変数ファイルを確認してみましょう。

```
[ansible@ansible lab4]$ vi external_vars.yml
```

```
---
epg: Web_EPG
bd: ACILab_BD1
ap: 3Tier_App
tenant: ACILab
```

3. ansible の Putty ターミナルで次のコマンドを入力し、VI エディタで ansible プレイブックを表示/編集します。

```
[ansible@ansible lab4]$ vi create_epg.yml
```

プレイブック全体を確認してその作用を理解してください。

注：使用する Ansible モジュール

aci_ap - https://docs.ansible.com/ansible/2.5/modules/aci_ap_module.html [英語]

aci_epg - https://docs.ansible.com/ansible/2.5/modules/aci_epg_module.html [英語]

```
---

- name: playbook for creating bd
  hosts: apic
  connection: local
  gather_facts: no
  vars_files:
    - external_vars.yml

  tasks:
  - name: create app network
    profile aci_ap:
      ap: "{{ ap }}"
      tenant: "{{ tenant }}"
      state: present
      hostname: "{{ inventory_hostname }}"
      username: "{{ user }}"
      password: "{{ pass }}"
      validate_certs: false

  - name: ensure web epg
    exists aci_epg:
```

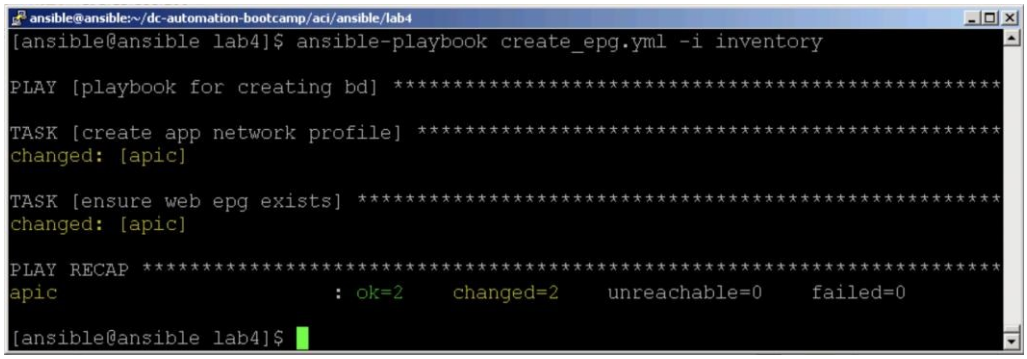
```
      epg: "{{ epg }}"
      bd: "{{ bd }}"
      ap: "{{ ap }}"
      tenant: "{{ tenant }}"
      state: present
      hostname: "{{ inventory_hostname }}"
      username: "{{ user }}"
      password: "{{ pass }}"
      validate_certs: false
```

4. プレイブックをひととおり確認したら、VI エディタを終了します。

5. Putty ターミナルで以下のコマンドを入力して、ansible プレイブックを実行します。

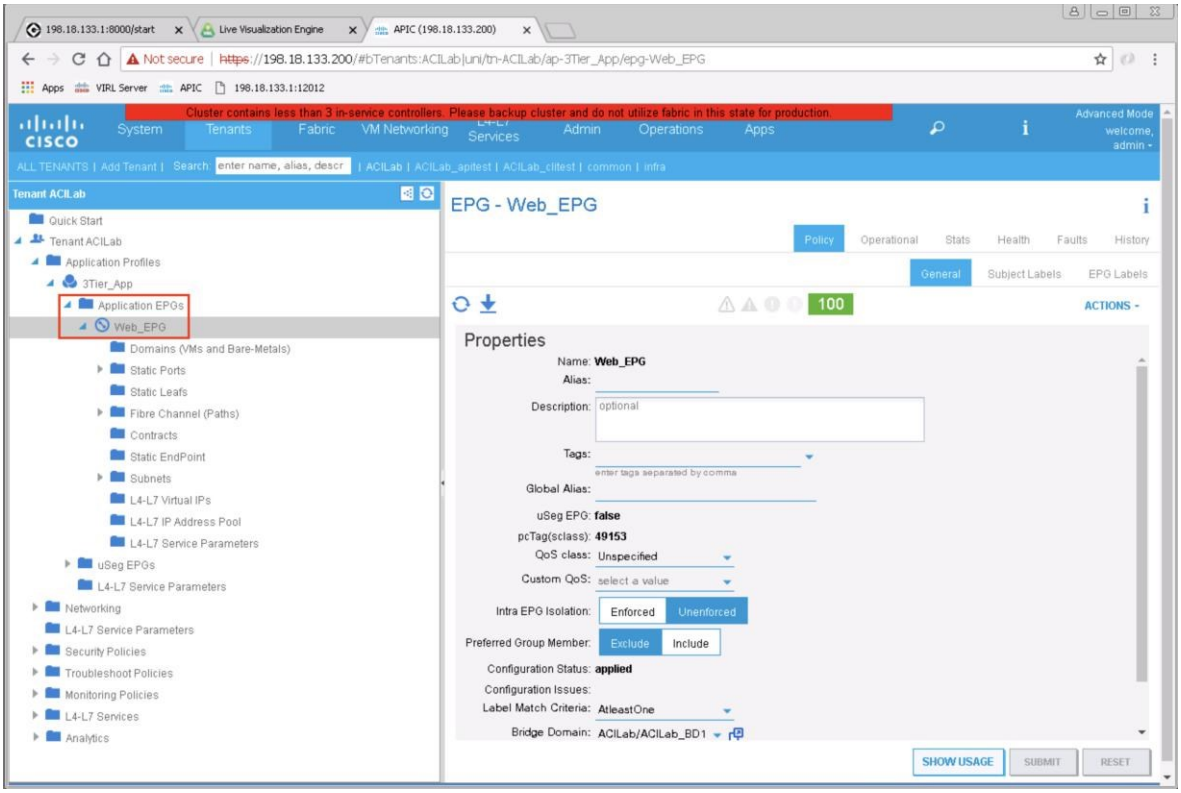
```
[ansible@ansible lab4]$ ansible-playbook create_epg.yml -i inventory
```

実行後、端末には次のように表示されます。



6. ここで、リモート デスクトップ システムの Web ブラウザを使用して、APIC コントローラの Web UI にアクセスできます。[テナント（Tenants）] タブに移動し、 *「ACILab」* テナントをダブルクリックします。ページの左側で、[アプリケーションプロファイル（Application Profiles）] > [3Tier_App] > [アプリケーション EPG（Application EPGs）] > [Web_EPG] の順にフォルダを開き、EPG が設定されていることを確認します。

図 16. 新しく作成されたアプリケーション プロファイルと EPG



これでシナリオ 2 のラボ 6 は完了です。

お問い合せ先

シスコシステムズ合同会社

〒107 - 6227 東京都港区赤坂9-7-1 ミッドタウン・タワー
http://www.cisco.com/jp