スクリプト化

ネットワークのプログラム可能性の利用

産業革命によって、商品の製造技術が近代化され、手作業による生産法から機械化された製造へと移り変わりました。手作業から自動化された操作への移行は、人の生産性を変化させ、機械によって簡単に実現できる繰り返し作業から人を解放できるようにしました。関連するコスト面での削減、高速化、品質向上により、より低いコストで、短時間に、より多くの作業が行えるようになり、高い品質の製品を生み出せるようになりました。プログラム可能性は、産業革命が商品にもたらしたものと同じ成果をネットワークに必ずもたらすものと考えられます。

IT 業界における自動化への必然的な移行により、さらに迅速に望ましい目標を達成し、需要に応じたタイムリーなやり方で迅速にインフラストラクチャをプロビジョニングし、コスト効率を高める方法を個人や企業に提供して、構成した結果の一貫性を高めています。ACI は、ネットワーク エンジニア、開発者、および初心者にまで自動化に向けてのアプローチ可能なパスを提供するため、一般的なツールや言語を使用し、固有の機能をプログラムで完全に公開することにより、これらすべての利点が利用できるようにしています。ACME は IT 組織における本来の DevOps に着手したばかりですが、これらの利点によってビジネスに後れを取ることはありません。

ACI で使用可能な総合的なプログラム可能性の機能により、誰もが恩恵を得られます。ACME のネットワーク エンジニアリング チームと設計チームは、大規模な構成の短時間でのプロビジョニングと、すべての可動部分を自動化する機能によって実現される一貫性による恩恵を受けることができます。運用チームは APIC 内に含まれている大量の情報を利用してプロセスを

合理化し、良質なメトリックを収集し、イベントをより正確に関連付け、解決までの時間を短縮して顧客満足度を引き上げることができます。

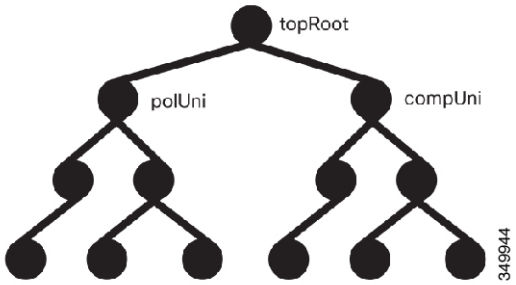
ACI とスクリプト化

ネットワークのプログラム可能性の目標は明確ですが、これらの目標を実現する手法の把握は困難でした。従来のネットワーキングデバイスは、人の目で確認するための出力を提供し、人が簡単に入力できるテキスト入力を使用して構成を行います。ただし、これらの目標は、自動化主導のアプローチとは対照を成しています。マシンは、何らかの構造化形式で提供されるデータをより簡単に処理できます。視覚的な訴えを持たない構造化データは高速で解析でき、また、包括的なオブジェクト指向の構成モデルを表すことが可能な詳細すべてを簡単に表現できます。

ACI は、詳しく文書化された REST API を使用して、使用およびポスト可能なアプリケーション ベースのセマンティックによるネットワーク構成を表す高度なオブジェクト モデルを使用します。オブジェクト モデルにこのインターフェイスを提供するだけでなく、ACI は、このデータの読み取りや操作を行うための、ユーザが安心してオープン スタンダードとオープン ソースを使用してプログラミングできるさまざまなレベルでのアクセス方法も数多く備えています。

オブジェクト モデルへの参照

図 *1:* オブジェクト モデルのトップ レベルの表現



オブジェクトモデルの包括的な概要はこのドキュメントの対象ではありませんが、プログラム可能性という観点から、ACI機能のあらゆる側面がオブジェクトモデルに含まれていることに注意することが重要です。つまり、ファブリックで行えるすべての構成は、REST API を使用してプログラムで行えるということです。これには、内部ファブリックのネットワーキング、外部のネットワーキング、仮想化統合、計算統合、およびその他すべての製品の側面が含まれています。

このデータは、管理情報ツリー内に、プロパティ、アイデンティティ、および適用する一貫性のルールによるプログラマチック オブジェクトとして表現されたモデルのあらゆる部分とともに保存されます。これにより、モデルの構成済みの状態が、古いノードやエントリによって収拾がつかなくなったりすることなく、あらゆる側面の検査や操作ができ、ユーザのニーズに応えられるようにすることができます。

プログラマチック インターフェイス

APIC は、構成を許可し、管理および運用可能な状態を提供するだけでなく、その構成を従属コンポーネントにまで拡張するという点で、柔軟性にかなり優れています。これらの機能を促進するインターフェイスのカテゴリは主に 2 つあります。ノースバウンド REST API とサウスバウンド プログラマチック インターフェイスです。

ノースバウンド REST API は構成を許可するとともに、コントローラに管理機能へのアクセスを提供します。このインターフェイスは、GUIやCLIの重要なコンポーネントであり、また、自動化ツール、プロビジョニング スクリプト、およびサード パーティのモニタリング ツールや管理ツールへのアクセス ポイントにもなっています。REST API は、構成変更を行うためのファブリックへの唯一のエントリ ポイントであり、一貫したプログラミングを行えるようにするための、アーキテクチャの重要な側面となっています。

APIC のサウスバウンド インターフェイスは、ファブリックを越えて従属するデバイスまでの拡張を目的とする宣言型モデルを考慮しています。つまり、ACIファブリックの開放性における重要な側面であり、そのポリシーにおいては、APIC を通じてプログラミングすれば、ハイパーバイザ、L4-7デバイス、将来的にはその他にもプッシュすることができつため、それらのデバイスを個々に構成する必要はありません。このサウスバウンドの拡張機能は、L4-7デバイス パッケージと OpFlex という 2 つのモデルを通じて実現されます。

L4-7 デバイス パッケージ インターフェイスでは、ACI ポリシーについての暗黙の知識を持たない既存の L4-7 デバイスに ACI がポリシーを適用できます。IP 経由でアクセス可能な何らかの形式のインターフェイスがデバイスにある限り、これらのデバイスはどのベンダーのものであってもかまいません。デバイス パッケージの実際の実装は、固有の構成インターフェイス、つまり、REST、CLI、SOAP などを介してデバイスに到達可能な、含まれている実行環境内の APIC 上で実行する、Python スクリプトで行われます。ユーザがサービス グラフや EPG ポリシーの変更を加えると、デバイス パッケージが APIC ポリシーを L4-7 デバイスの API コールに変換します。

OpFlexは、情報モデルの一部として定義された管理対象オブジェクトのデータ交換を可能にするように設計されています。OpFlex 自体は、情報モデルを管理せず、ツリーの各ノードに関連付けられたユニバーサル リソース識別子（URI）を持つツリー ベースの抽象モデルと一緒に使用できます。プロトコルは、XMLおよびJSON（および一部のシナリオで使用されているバイナリ符号化）をサポートし、JSON-RPC over TCP などの標準的なリモート プロシージャ コール（RPC）メカニズムを使用するように設計されています。現在、ACIでは、ポリシーをアプリケーション仮想スイッチに拡張したり、OpenStack にグループ ベースのポリシーを拡張するために、OpFlexが使用されています。

REST API について

Application Policy Infrastructure Controller (APIC) REST API は、REST アーキテクチャを使用するプログラム的なインターフェイスです。API は JavaScript オブジェクトの表記（JSON）または拡張マークアップ言語（XML）のドキュメントを含む HTTP（デフォルトでは無効）または HTTPS のメッセージを受け入れ、返します。プログラミング言語を使用して、API メソッドまたは MO の説明を含むメッセージや JSON または XML ドキュメントを生成できます。

REST API は、MIT へのインターフェイスであり、オブジェクト モデルの状態を操作できます。APIC コマンドライン インターフェイス（CLI）、GUI、および SDK は同じ REST インターフェイスを使用するため、情報を表示する場合は常に、REST API を介して読み込まれ、構成変更が行われた場合は REST API を通じて書き込まれます。また、REST API は、統計情報、フォールト、監査イベントなど、他の情報を取得できるインターフェイスも提供し、さらにプッシュベースのイベント通知をサブスクライブする手段も提供するため、MIT 内で変更が発生した場合はイベントを Web ソケットを通じて送信できます。

API では、HTTP を通じた POST 操作、GET 操作、DELETE 操作など、標準的な REST メソッドがサポートされています。POST メソッドと DELETE メソッドは、同じ入力パラメータで複数回呼び出されても、それ以上の効果を持たないべき等です。GET メソッドはべきゼロで、何らの変更も行うことなく（つまり、読み取り専用操作）ゼロ回または複数回呼び出すことができます。

REST インターフェイスに出入りするペイロードは、XML エンコーディングまたは JSON エンコーディングによりカプセル化できます。XML の場合、エンコーディング操作は簡単です。要素タグはパッケージとクラスの名前で、そのオブジェクトのプロパティはその要素の属性として指定します。含有は、子要素を作成して定義します。

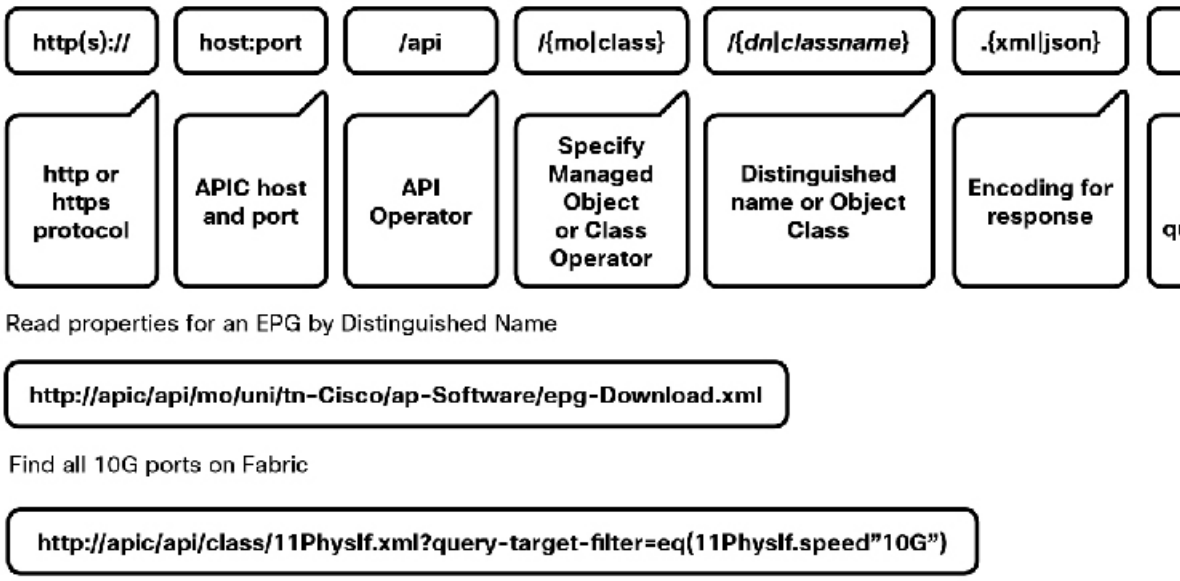
JSON の場合、エンコーディングにはツリーベースの階層を反映する特定のエントリの定義が必要です。ただし、その定義はツリーのすべてのレベルで繰り返されるため、最初に理解していれば実装はかなり簡単です。

- すべてのオブジェクトを JSON ディクショナリとして記述します。この場合、キーはパッケージとクラスの名前で、値は、属性と子という 2 つのキーを持つ別のネストされたディクショナリになります。
- 属性キーには、オブジェクト上の属性を定義するキー値ペアを記述する、さらにネストされたディクショナリが含まれています。
- 子キーには、すべての子オブジェクトを定義するリストが含まれています。このリストの子オブジェクトは、ここで説明したように定義された、ネストされたオブジェクトを含むディクショナリです。

読み取り操作

オブジェクト ペイロードは、XML または JSON として適切に符号化された後は、REST API での作成、読み取り、更新、削除の操作に使用できます。次の図に、REST API からの読み取り操作の構文を示します。

図 *2 : REST* 構文



REST API は、HTTP ベースであるため、特定のリソース タイプにアクセスするようにユニバーサル リソース識別子（URI）を定義することが重要です。要求 URI の最初の 2 つのセクションでは、APIC のプロトコルとアクセスの詳細を単に定義します。要求 URI の次のセクションは、API の呼び出しを指示するリテラル文字列 **/api** です。先に説明したように、読み取り操作は一般に、オブジェクトまたはクラスに対するものであるため、URI の次の部分で操作が MO に対してか、クラスに対してかを指定します。次のコンポーネントで、オブジェクトベースのクエリに対して照会する完全修飾 Dn か、クラスベースのクエリに対するパッケージとクラス名のいずれかを定義します。要求 URI の最後の必須部分はエンコーディング形式で、.xml か .json のいずれかです。これはペイロード形式を定義する唯一の方法です（APIC は Content-Type やその他のヘッダーを無視します）。

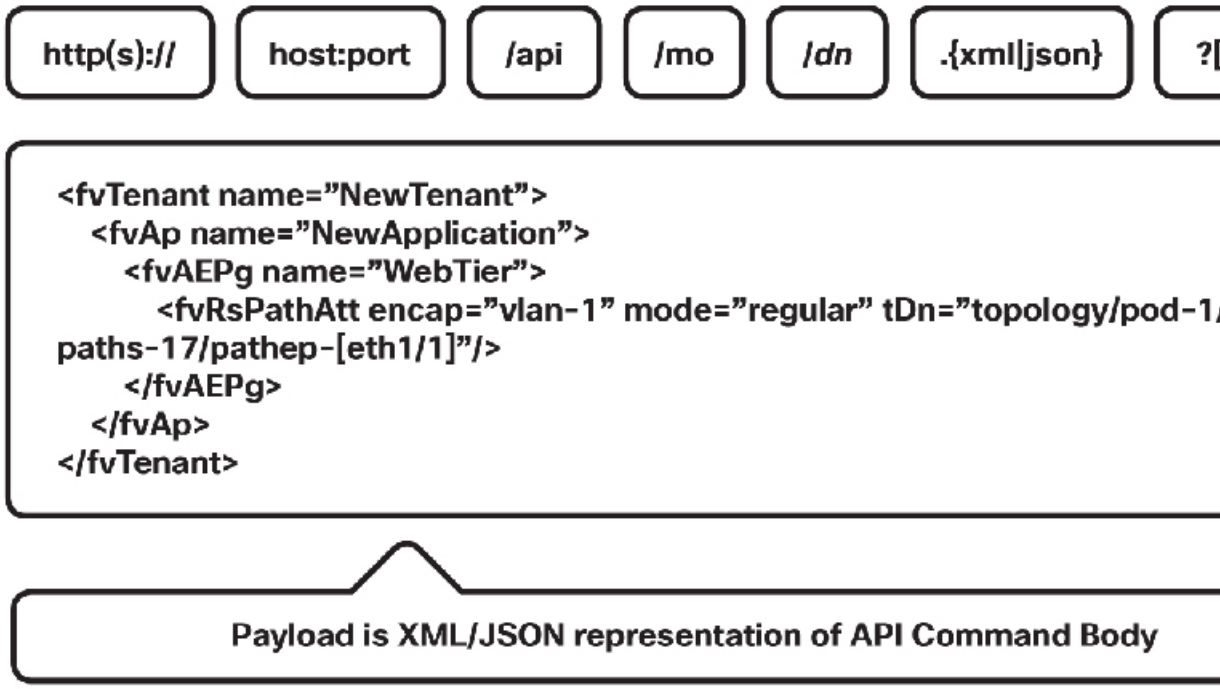
書き込み操作

REST API の作成操作と更新操作は両方とも POST メソッドを使用して実装されます。そのため、まだない場合はオブジェクトが作成され、既に存在する場合は、既存の状態と必要な状態との間の変化を反映するように更新されます。

作成操作と更新操作の両方に複合オブジェクト階層が含まれていることがあります。そのため、すべてのオブジェクトが同じコンテキスト ルートにあり、REST API のデータ ペイロードの 1 MB の制限を超えていない限り、1 つのコマンドで完全なツリーを定義することができます。この制限は、パフォーマンスを保証し、高負荷時にシステムを保護するために設定されています。

コンテキスト ルートは、APIC が複数のコントローラに情報を配信する方法を定義するのに便利で、一貫性の確保に役立ちます。ほとんどの場合、構成はユーザに透過的である必要があります。ただし、かなり大規模な構成で、それが分散トランザクションになる場合は、小さく分割する必要がある場合があります。

図 *3 : REST* ペイロード



作成操作と更新操作は、オブジェクト レベルで常に対象となること以外は、読み取り操作と同じ構文を使用します。これは、特定のクラスのあらゆるオブジェクトに変更を加えることができないからです（その必要もありません）。作成操作または更新操作は、特定の管理対象オブジェクトを対象とする必要があります。そのため、リテラル文字列 **/mo** で管理対象オブジェクトの Dn が提供されることを示し、その次に実際の Dn を示します。フィルタ文字列を POST 操作に適用できます。たとえば、POST 操作の結果を応答で取得する場合は、**rsp-subtree=modified** クエリ文字列を渡し、POST 操作によって変更されたオブジェクトを応答に含めるように指示します。

POST 操作のペイロードには、Cisco API コマンド本文を定義する管理対象オブジェクトを表す XML または JSON の符号化データが含まれます。

認証

REST API のユーザ名ベースおよびパスワードベースの認証は、POST 操作の Dn ターゲットとして、**aaaLogin**、**aaaLogout**、および **aaaRefresh** などの特殊な要求 URI のサブセットを使用します。ペイロードには、シンプルな XML ペイロードまたは JSON ペイロードが含まれ、これらには **aaaUser** オブジェクトの MO 表現と、ユーザ名とパスワードを定義する属性名および **pwd** が含まれています。たとえば、**<aaaUser name='admin' pwd='insieme'/>** のようになります。POST 操作の応答には、Set-Cookie ヘッダーと、応答の名前付きトークンの **aaaLogin** オブジェクトの属性の両方として認証トークンが含まれます。この場合の XPath はエンコーディングが XML の場合は **/imdata/aaaLogin/@token** です。REST API の後続の操作では、このトークン値を **APIC-cookie** という名前の cookie としてその後の要求の認証に使用できます。

Filters

REST API は、柔軟性のあるさまざまなフィルタをサポートしており、検索範囲を絞り込み、よりすばやく情報を見つけるのに役立ちます。フィルタ自体は、クエリのURIオプションとして付加され、疑問符（?）で始まり、アンパサンド（&）で連結されます。複数の条件をまとめて連結して複雑なフィルタを形成できます。

次のクエリ フィルタを使用できます。

図 *4:* クエリ フィルタ

Filter type	Syntax	Cobra Query Property	Descriptio
query-target	{self | children | subtree}	AbstractQuery.queryTarget	Define the
target-subtree-class	<class name>	AbstractQuery.classFilter	Respond c including s
query-target-filter	<filter expressions>	AbstractQuery.propFilter	Respond c matching
rsp-subtree	{no | children | full}	AbstractQuery.subtree	specifies c included in
rsp-subtree-class	<class name>	AbstractQuery.subtreeClassFilter	Respond c classes
rsp-subtree-filter	<filter expressions>	AbstractQuery.subtreePropFilter	Respond c matching
rsp-subtree-include	{faults | health :stats : ...}	AbstractQuery.subtreeInclude	Request a
order-by	<classname.property>| {asc | desc}	*NotImplemented*	Sort the re on the pro

Query filters

サブスクリプション

REST API は、アクティブな API セッション中の 1 つ以上の MO へのサブスクリプションをサポートします。ユーザまたはシステムにより開始されたアクションによって、MO が作成、変更、または削除されると、イベントが生成されます。サブスクライブされたアクティブなクエリ上のデータがイベントにより変更されると、APICはそのサブスクリプションを作成したAPIクライアントに通知を送信します。

API インスペクタ

GUI で実行されたすべての操作が REST コールを呼び出し、アクセスする情報をフェッチし、コミットします。API インスペクタは、URI やペイロードをリアルタイムで表示することに

よって、GUI を移動しながら REST インターフェイスで行われていることを調査するプロセスをさらに簡略化します。新しい構成がコミットされると、API インスペクタにはその結果の POST 要求が表示され、GUI に情報が表示されると、GET 要求が表示されます。

API インスペクタを起動するには、Cisco APIC の GUI の右上に表示されている [Account] メニューからアクセスします。[Welcome, <username>] をクリックし、[Show API Inspector] オプションを選択します。

API インスペクタが起動したら、REST メソッド、URI、およびペイロードとともにタイム スタンプが表示されます。画面に表示されるデータのサブスクリプションを GUI が更新するときにリストが更新されることがあります。

上記の出力から、最後にログに記録した項目には POST 要求とともに、**Cisco** と言う名前のテナントと、オブジェクトに定義されたいくつかの属性を含む JSON ペイロードがあることがわかります。

```
url: http://172.16.176.176/api/node/mo/uni/tn-Cisco.json


{
"fvTenant": {
"attributes": {
"name": "Cisco",
"status": "created"
},
"children": [
{
"fvBD": {
"attributes": {
"mac": "00:22:BD:F8:19:FF",
"name": "CiscoBd",
"status": "created"
},
"children": [
{
"fvRsCtx": {
"attributes": {
"tnFvCtxName": "CiscoVrf",
"status": "created,modified"
},
"children": []
}
}
]
}
},
{
"fvCtx": {
"attributes": {
"name": "CiscoVrf",
"status": "created"
},
"children": []
}
}
]
}
}
```

開発テクニック

ACIには、プログラミングやプログラマティックインターフェイスとの対話が安心してできるレベルが異なるエンジニアが使用可能な開発コード用の方法が数多く用意されています。

最も基本的で、単純なテクニックでは、APIインスペクタ、Visoreが収集するかXML/JSONをGUIから保存することによって情報を取得し、たとえばPOSTmanなどの一般的に無償で提供されているツールを使用して、この情報をREST APIに送信します。

このメソッドを設定することによって、ユーザは一般的な用語を使用し、また、ネットワーキング構成の深い理解と、それらをACIのポリシー言語ならびに良く使用されているPythonプログラム言語の能力や柔軟性と結び付け、プログラマチックなやり方でACIを構成することができます。ACIツールキットは、最も一般的なACI構築ブロックを公開するオープンソースで開発されたユーティリティであり、ユーザはテナント、アプリケーションプロファイル、EPG、および関連付けられた概念をすばやく作成して、それらを物理インターフェイスに接続できます。提供される合理化されたインターフェイスは、非常にすばやく導入でき、ユーザはアプリケーションの開発にすぐに着手できます。

使用可能な開発ツールの中で最もパワフルなのはCobra SDKです。使用可能なACIオブジェクトモデルの詳細な表現、包括的なデータ検証、クエリおよびフィルタリングの広範なサポートにより、Corbaは開発者にもユーザにも、同様なACIの完全なエクスペリエンスを保証します。

POSTman

POSTmanは、Chrome Webブラウザのオープン ソースの拡張機能で、REST クライアント機能を使い勝手の良いパッケージとして提供します。POSTmanは、構成、操作、ポリシー、運用状態データなどを表すデータを送受信するためにAPIC REST インターフェイスとの対話に使用できます。RESTの構造に不慣れなユーザの場合も、API インスペクタを使用すると、特定の操作についてどんな基盤コールがGUIに行われているかが表示されるため、それらをキャプチャし、POSTmanを使用してそれらの操作を簡単に再生できます。さらに、POSTmanでは、要求を変更できます。GUI操作は1回で行え、キャプチャしたデータの属性を変更し、REST APIに送信し直してから変更が行われます。

インストール

POSTmanを起動するには、まず、http://www.getpostman.com で入手可能なChrome Webブラウザのプラグインをダウンロードします。プラグインをインストールすると、Chromeアプリケーション ランチャを使用してアクセスできるようになります。

最初、2つの主要セクション、サイドバー（左側）と要求コンストラクタ（右側）があるインターフェイスがユーザに表示されます。サイドバーを使用して、POSTmanが送信したREST要求の履歴と、共通タスクを含んでいる要求のコレクションとを切り替えます。

Collections

コレクションで作成する便利なポストは、基本的なログイン操作です。これを行うには、ユーザはまず、サイドバーの [Collections] タブをクリックする必要があります。サイドバー内では、プラス（+）が付いた小さなフォルダが表示されます。それをクリックすると、その時点でポップアップが表示され、ユーザはコレクションに名前を付けるように求められます。この例では、コレクションを「APIC」という名前にし、[Create] ボタンをクリックします。

ログイン要求の構築

ここで、新しい要求を構築できます。要求コンストラクタで、[Enter request URL here] が表示されたら、次の要求 URI を入力して、APICIPADDRESS を APIC の IP に置き換えます。

```
https://apicipaddress/api/mo/aaaLogin.xml
```

この要求 URI は REST API のログイン メソッドを呼び出します。ログインにはポスティングデータが必要であるため、HTTP メソッドを変更する必要があります。これを行うには、要求 URL の右にあるドロップダウンリストをクリックします。デフォルトでは GET 要求ですが、ドロップダウンリストから POST を選択する必要があります。

これで、選択した POST メソッドで、REST ペイロードを提供できるようになります。データを REST 経由で送信する場合は、「raw」要求本文セレクタを選択する必要があります。

これで、要求のペイロードを入力できるようになります。これは、認証に使用されるユーザ名とパスワードを含むシンプルな XML です。URL は https です。つまり、Web ブラウザと APIC 間で暗号化されるため、クリアテキストで送信されるデータはありません。次の要求本文を入力し、USERNAME と PASSWORD を正しいユーザ名とパスワードに置換します。

```
<aaaUser name='USERNAME' pwd='PASSWORD'/>
```

この要求を構築すると、要求を送信できるようになりますが、これは一般的に使用されている方法であるため、要求をコレクションに追加する必要があります。これを行うには、要求本文の下にある [Add to collection] ボタンをクリックします。既存のコレクション リストから「APIC」コレクションを選択し、要求名を「Login」に変更してから、[Add to collection] をクリックします。

要求をコレクションに追加することによって、必要に応じ、APIC とのログイン セッションを確立するために後ですばやくアクセスできるようになります。

上記のステップが完了したら、要求の送信準備が整ったことになります。要求コンストラクタの [Send] ボタンをクリックすると、APIC とのログイン セッションを示す XML を REST API が返します。POSTman GUI には次のように表示されます。

APIC への照会

構築する次の要求は、システム上のテナントのリストを APIC に照会する要求です。まず、要求コンストラクタの [Reset] ボタンをクリックし、上記と同じ手順に進みます。ただし、要求 URL は次のように表示されます。

```
https://apicipaddress/api/class/fvTenant.xml
```

要求メソッドは GET に変更されます。

[Add to collection] をクリックし、APIC コレクションに要求を配置して、名前に [Query APIC for tenants] と入力します。

ここで [Send] をクリックすると、この要求は、XML で符号化されたテナントのリストを右側のコンストラクタ ペインの応答本文セクションに返します。

APIC での構成変更

構成の変更を行うにはログインと同様な POST 要求を使用しますが、要求 URL と本文には別の情報を含めます。

この例では、新しいテナントがファブリックに作成されます。要求コンストラクタの [Reset] ボタンをクリックして既存の要求フィールドをすべてクリアし、次の URL を使用します。

```
https://apicipaddress/api/mo/uni.xml
```

メソッドを POST に変更します。

要求ペイロードで、次のデータを入力します。

```
<fvTenant name="Cisco"/>
```

要求 URL は、このクエリのターゲットがテナントが存在するポリシー ユニバースであることを指定します。このターゲットの範囲を正しく指定することで、テナントを表すデータがペイロードに表示されます。この場合は、Cisco という名前のテナントが作成されます。

クエリ ガイダンスとしての API インスペクタの使用

スクリプト化の項の概要で説明したように、API インスペクタはカスタム REST 要求を構築するためのガイドラインとして使用できます。その項の例で進めると、要求 URL は次のようになります。

```
https://apicipaddress/api/node/mo/uni/tn-Cisco.json
```

ペイロードは、次のように、JSON の圧縮バージョンになります。

```
 {"fvTenant": {"attributes": {"name": "Cisco", "status": "created"},
"children": [{"fvBD": {"attributes": {"mac": "00:22:BD:F8:19:FF",
"name": "CiscoBd", "status": "created"}, "children": [{"fvRsCtx":
{"attributes": {"tnFvCtxName": "CiscoVrf", "status":
"created,modified"}, "children": [] } } ] } }, {"fvCtx": {"attributes":
{"name": "CiscoVrf", "status": "created"}, "children": [] } } ] } }
```

この要求 URI とペイロードを変更し、テナント名の「Cisco」を別の名前に置換し、まったく新しいテナントを同じ構成で作成することができます。新しい要求 URL および JSON は次のようになります。

```
https://apicipaddress/api/node/mo/uni/tn-Acme.json
```

```
 {"fvTenant": {"attributes": {"name": "Acme", "status": "created"},
"children": [{"fvBD": {"attributes": {"mac": "00:22:BD:F8:19:FF",
"name": "AcmeBd", "status": "created"}, "children": [{"fvRsCtx":
{"attributes": {"tnFvCtxName": "AcmeVrf", "status": "created,modified"},
 "children": [] } } ] } }, {"fvCtx": {"attributes": {"name": "AcmeVrf",
"status": "created"}, "children": [] } } ] } }
```

これらの値を POSTman の POST 要求に配置し、保存したログイン要求を使用してログイン セッションを確立した後に、前に作成した Cisco というテナントと同じ新しいテナントである「Acme」を作成できます。GUI を手動でクリックしたり、他の手作業は必要ありません。

Cobra SDK と Arya

完全な Cisco ACI Python SDK を Cobra と言います。これは、API の Python の本来の実装であり、REST のすべての機能固有のバインディングを提供するとともに、オブジェクト モデルの完全なコピーを備えているため、データの整合性が確保されるだけでなく、ACI で使用可能な機能と関数をすべてサポートします。Cobra はルックアップやクエリ、GUI が使用する REST に一致し、API インスペクタを使用して検出可能なオブジェクトの作成、変更、および削除を実行するメソッドを提供します。その結果、GUI で作成したポリシーをプログラミング テンプレートとして使用して、開発を促進することができます。

Cobra のインストール プロセスは簡単で、標準的な Python 配布ユーティリティを使用できます。Cobra は APIC 上に .egg ファイルとして配布され、easy_install を使用してインストールできます。また、http://github.com/datacenter/cobra の github でも入手できます。

Cobra SDK の完全なドキュメントは、http://cobra.readthedocs.org/en/latest/ で入手できます。

セッションの確立

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

```
import cobra.mit.access
import cobra.mit.session
apicUri = 'https://10.0.0.2'
apicUser = 'username'
apicPassword = 'password'
ls = cobra.mit.session.LoginSession(apicUri, apicUser, apicPassword)
md = cobra.mit.access.MoDirectory(ls)
md.login()
```

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

オブジェクトの使用

Cobra SDK を使用して、このワークフローに概ね従っている MIT を操作します。

1. 操作するオブジェクトを特定します。
2. 属性を変更する、あるいは子オブジェクトを追加または削除する要求を構築します。
3. オブジェクトに加えた変更をコミットします。

たとえば、新しいテナントを作成する場合は、まず、MITのどこにテナントが配置されるか、この場合は、**policy Universe**の監視対象オブジェクトの子オブジェクト（**polUniMo**）を特定します。

```
import cobra.model.pol
polUniMo = cobra.model.pol.Uni('')
```

polUniMo オブジェクトを定義すると、テナント オブジェクトを **polUniMo** の子として作成できます。

```
import cobra.model.fv
tenantMo = cobra.model.fv.Tenant(polUniMo, 'cisco')
```

これらのすべての操作によって作成されるのは、Pythonオブジェクトだけです。構成を適用するには、その構成をコミットする必要があります。これを行うには、**ConfigRequest** というオブジェクトを使用します。**ConfigRequest** は、単一コンテキストに分類される MO ベースのクラスのコンテナとして機能し、単一のアトミック POST 操作にすべてコミットできます。

```
import cobra.mit.request
config = cobra.mit.request.ConfigRequest()
config.addMo(tenantMo)
md.commit(config)
```

ConfigRequestオブジェクトが作成された後、**tenantMo**オブジェクトが要求に追加されたら、その構成を **MoDirectory** オブジェクトを使用してコミットします。

先の例では、最初のステップで **polUni** オブジェクトのローカル コピーを構築します。命名プロパティがないため（二重の空の単一引用符によって反映されている）、MITでルックアップしてオブジェクトの完全なDnを把握する必要がありません。常に**uni**として知られています。

MITにより詳細なものをポストする場合で、オブジェクトに命名プロパティがあるときは、そのオブジェクトのルックアップを実行する必要があります。たとえば、構成を既存のテナントにポストする場合は、そのテナントを照会し、その下にオブジェクトを作成します。

```
tenantMo = md.lookupByClass('fvTenant', propFilter='eq(fvTenant.name, "cisco")')
tenantMo = tenantMo[0] if tenantMo else None
```

その結果の**tenantMo**オブジェクトのクラスは**cobra.model.fv.Tenant**となり、.dn、.status、.nameなど、オブジェクト自体をすべて説明するプロパティが含まれます。**lookupByClass()** エントリはアレイを返します。これは、複数のオブジェクトを返すことができるためです。この場合、コマンドは限定的であり、特定の名前を使用して **fvTenant** オブジェクトでフィルタリングします。テナントの場合、name 属性は命名属性と呼ばれる特殊なタイプの属性です。命名属性は、ローカルの名前空間で一意である必要がある相対名の構築に使用されます。その結果、名前に関するフィルタを使用した **fvTenant** での **lookupByClass** は、長さ 1 または None のアレイ、つまり何も検出されなかったことを返します。

ルックアップを全面的に避けるには、Dnオブジェクトを構築し、そのDnの子オブジェクトを作成できます。この方法は、親オブジェクトが存在する場合にだけ有効です。

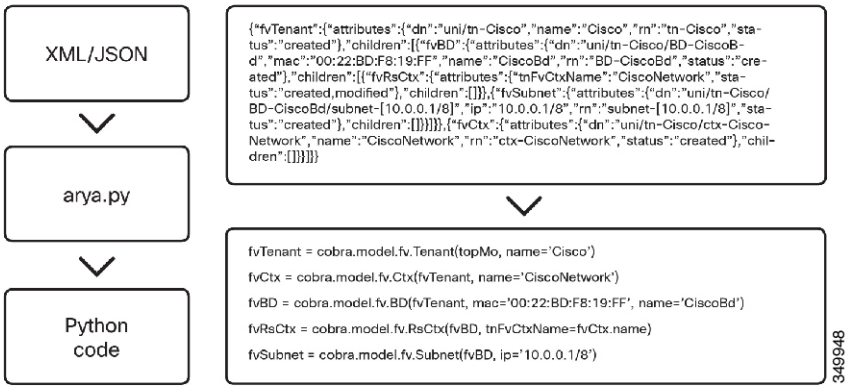
```
topDn = cobra.mit.naming.Dn.fromString('uni/tn-cisco')
fvAp = cobra.model.fv.Ap(topMo, name='AppProfile')
```

Cobra と対話するためのこのような基本的な方法によって、ネットワーク構成の自動化、トラブルシューティングの実行、およびネットワークの管理に役立てることができる、より複雑なワークフローの作成に必要な構成要素が提供されます。

Python アダプタへの Cisco APIC REST

要求の構築プロセスには時間がかかります。これは、オブジェクトデータのペイロードを、加えたいオブジェクト変更を反映した Python コードとして表す必要があるためです。Cobra SDK が Cisco ACI オブジェクトで直接モデル化されるため、オブジェクト モデルに存在するものから直接コードを生成できる必要があります。当然、これは、シスコ アドバンスド サービスが開発したツールを使用して実行できます。ツールは Arya と呼ばれる Python アダプタへの Cisco APIC REST です。

図 *5 : Python* アダプタへのサンプル *REST*



上の図には、API インスペクタ、Visore からの可能性がある入力や、REST クエリの出力が Cobra SDK コードにいかに迅速に変換され、トークン化され、より高度な方法で再利用されるかが明確に示されています。

Arya のインストールは比較的簡単で、ツールには外部依存関係がほとんどありません。Arya をインストールするには、Python 2.7.5 と Git がインストールされている必要があります。次のクイック インストール ステップを使用して Arya をインストールし、システムの Python に配置します。

```
git clone https://github.com/datacenter/ACI.git
cd ACI/arya
sudo python setup.py install
```

Arya をインストールしたら、Cisco ACI モデル化オブジェクトを表す XML または JSON を取得すると、Python コードにすばやく変換できます。たとえば、次のように入力します。

```
arya.py -f /home/palesiak/simpletenant.xml
```

このエントリによって次の Python コードが得られます。

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

```
'''
Autogenerated code using arya.py
Original Object Document Input:
<fvTenant name='bob'/>
''' raise RuntimeError('Please review the auto generated code before ' +
 'executing the output. Some placeholders will ' +
 'need to be changed')
# list of packages that should be imported for this code to work
import cobra.mit.access
import cobra.mit.session
import cobra.mit.request
import cobra.model.fv
import cobra.model.pol
from cobra.internal.codec.xmlcodec import toXMLStr
# log into an APIC and create a directory object
ls = cobra.mit.session.LoginSession('https://1.1.1.1', 'admin', 'password')
md = cobra.mit.access.MoDirectory(ls)
md.login()
# the top level object on which operations will be made
topMo = cobra.model.pol.Uni('')
# build the request using cobra syntax
fvTenant = cobra.model.fv.Tenant(topMo, name='bob')
# commit the generated code to APIC
print toXMLStr(topMo)
c = cobra.mit.request.ConfigRequest()
c.addMo(topMo)
md.commit(c)
```

このコードを実行する前に、まず、ランタイムエラーが発生しているプレースホルダを削除する必要があります。これは、更新する必要がある他のトークン化された値が正しいことを確認するために、意図的に配置されています。たとえば、Cisco APIC の IP アドレスはデフォルトでは 1.1.1.1 ですが、これを実際の Cisco APIC の IP アドレスを反映するように更新する必要があります。クレデンシャルやその他のプレースホルダも同様です。

完全修飾階層を持たない入力 XML または JSON を提供すると、Arya はヒューリスティックからそれを特定できない場合があることを注意してください。この場合、プレースホルダにはテキストの **REPLACEME** が読み込まれるため、これを正しい Dn に置き換える必要があります。この Dn は、Visore のオブジェクトを照会するか、API インスペクタに表示されたオブジェクトの要求 URI を検査することによって見つけることができます。

ACI ツールキット

完全な Cisco Application Centric Infrastructure (ACI) オブジェクト モデルには多くのエントリが含まれているため、ネットワークのプログラム可能性を初めて知ったユーザには手ごわいかもしれません。ACI ツールキットは、ACI の概念を紹介し、共通のタスクやワークフローを迅速に起動させる手段をユーザに提供する、簡略化されたモデルのサブセットを利用できるようにします。さらに、多くのアプリケーションが ACI ツールキットの上部に構築されています。

ACI ツールキットの完全なドキュメントは、http://datacenter.github.io/acitoolkit/ で入手できます。

ACI ツールキットは、オペレータがすぐに使える便利なツールをいくつか提供しますが、本来の価値は、これらの例を開始点として見なし、特定のニーズに合うように、これらの例を変更

または拡張する機能にあります。試してみてください。自身の取り組みをコミュニティと共有してください。

ACI ツールキットのアプリケーション

エンドポイント トラッカー

エンドポイント トラッカー アプリケーションは、エンドポイント クラス（fvCEp）のサブスクリプションを作成し、ファブリック（サーバ、ファイアウォール、ロードバランサ、およびその他のデバイスなど）の各エンドポイントの関連詳細情報を MySQL データベースに読み込みます。MySQL のインストールはこのドキュメントの対象外であるため、MySQL サーバに新しいデータベースを作成するためのアクセス権がユーザにあると想定します。エンドポイント トラッカー アプリケーションには主要コンポーネントが 2 つあり、どちらも Python スクリプトです。

- **aci-endpoint-tracker.py**：このスクリプトは、エンドポイント クラスのサブスクリプションを作成し、MySQL データベースに読み込みます。
- **aci-endpoint-tracker-gui.py**：このスクリプトは、データベースのコンテンツをオペレータに表示する方法を提供する Web インターフェイスを作成します。次に例を示します。

図 *6:* エンドポイント トラッカー

Mac	IP	Tenant	App	EPG	Interface	Time Start	Time Stop
00:0A:F7:00:C5:E5	192.168.2.32	mgmt	mgmt-applications	server-management	opk2r16-5k01	2015-02-13 09:13:24	0000-00-00 00:00:00
00:0A:F7:00:C5:E5	192.168.2.32	mgmt	mgmt-applications	server-management	opk2r16-5k01	2015-02-13 09:13:24	0000-00-00 00:00:00
00:0C:29:1C:EB:7C	192.168.2.45	mgmt	mgmt-applications	server-management	opk2r16-5k01	2015-02-13 09:28:39	0000-00-00 00:00:00
00:0C:29:1C:EB:7C	192.168.2.45	mgmt	mgmt-applications	server-management	opk2r16-5k01	2015-02-13 09:28:39	0000-00-00 00:00:00
00:0C:29:D5:4A:AE	192.168.2.46	mgmt	mgmt-applications	server-management	opk2r16-5k01	2015-02-13 09:16:51	0000-00-00 00:00:00
00:0C:29:D5:4A:AE	192.168.2.46	mgmt	mgmt-applications	server-management	opk2r16-5k01	2015-02-13 09:16:51	0000-00-00 00:00:00
00:25:B5:00:00:08	0.0.0.0	infra	access	default	eth 1/101/1/5	2015-02-25 12:20:40	0000-00-00 00:00:00
00:25:B5:20:01:03	192.168.2.140	mgmt	mgmt-applications	server-management	opk2r15-ucs02b	2015-02-13 09:13:32	0000-00-00 00:00:00
00:25:B5:20:01:03	192.168.2.140	mgmt	mgmt-applications	server-management	opk2r15-ucs02b	2015-02-13 09:13:32	0000-00-00 00:00:00
00:25:B5:20:01:07	192.168.2.141	mgmt	mgmt-applications	server-management	opk2r15-ucs02b	2015-02-13 09:13:32	0000-00-00 00:00:00
Mac	IP	Tenant	App	EPG	Interface	Time Start	Time Stop

Sample Endpoint Tracker GUI

349949

エンドポイント トラッカーを起動するには、次の Python スクリプトを実行します。最初のスクリプトの aci-endpoint-tracker.py は、APIC に実際に接続し、データベースに読み込みます。2 番目のスクリプトは、理解可能な Web UI でコンテンツを表示できるようにします。

```
user@linuxhost:~/acitoolkit/applications/endpointtracker$ ./aci-endpoint-tracker.py
MySQL IP address: 127.0.0.1
MySQL login username: root
MySQL Password:
user@linuxhost::~/acitoolkit/applications/endpointtracker$ python aci-endpointtracker-
gui.py
MySQL IP address: 127.0.0.1
MySQL login username: root
MySQL Password:
* Running on http://127.0.0.1:5000/
* Restarting with reloader
```

これらの Python スクリプトを実行した後で、ブラウザを起動して、Web UI に進むことができます。ACI エンドポイント トラッカーの使用は、IP や MAC アドレスを検索フィールドに入力するようなもので、テーブルはそれに応じてフィルタリングされます。次の例では、IP アドレス 192.168.5.20 が検索フィールドに入力され、一致する結果が表示されています。

Mac	IP	Tenant	App	EPG	Interface	Time Start	Time Stop
00:50:56:A9:73:01	192.168.5.20	mynewproject	app1	db-epg	topology/pod-1/path-grp-[192.168.1.221]	2015-02-19 12:47:30	0000-00-00 00:00:00
Mac	IP	Tenant	App	EPG	Interface	Time Start	Time Stop

349950

エンドポイント トラッカー アプリケーションの興味深い使い方の 1 つに、テナント、アプリケーション、EPG など、他のファブリック構造にさまざまなエンドポイントがどのようにマッピングされているかを表示できる一連の可視化があります。

次に、いくつかのスクリンショットの例を示します。これらは、エンドポイントが ACI ファブリック内のどこにあり、それらが環境内の他のオブジェクトにどのように関係または依存しているかを示しています。

図 *7:* エンドポイントの分散の円グラフ ビュー

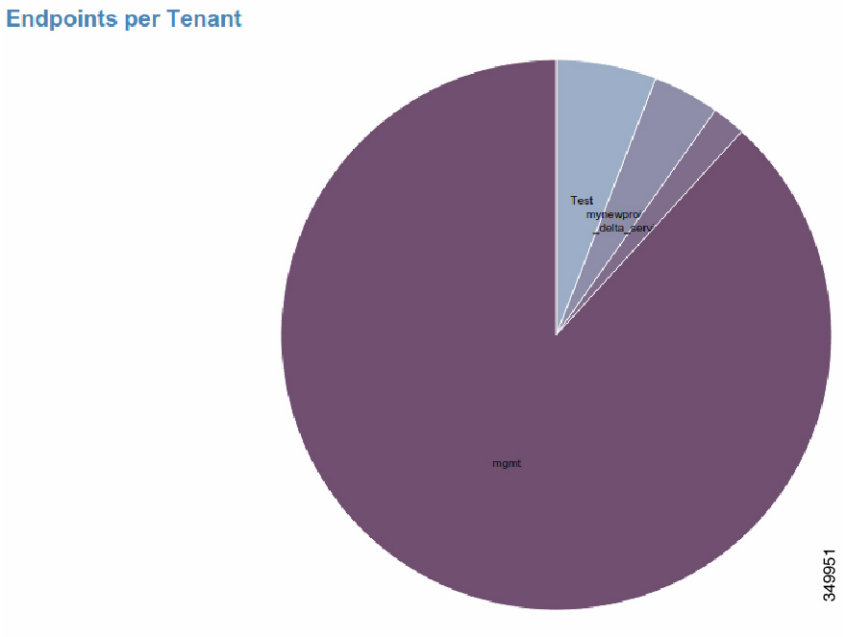


図 *8:* エンドポイントの関係性のツリー ビュー

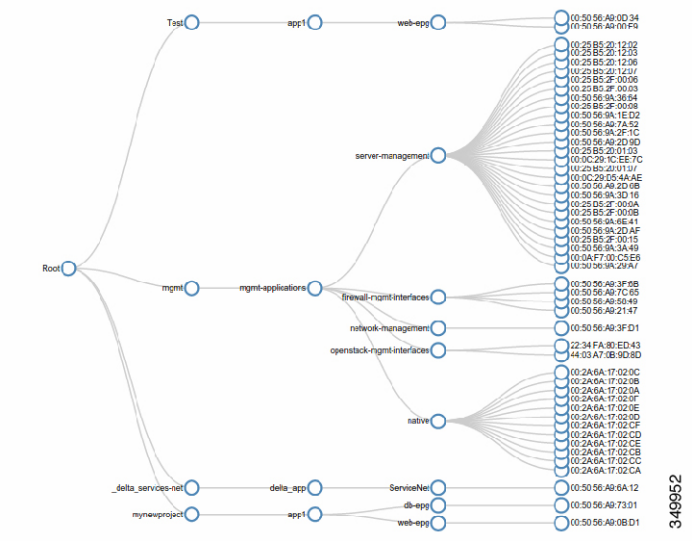
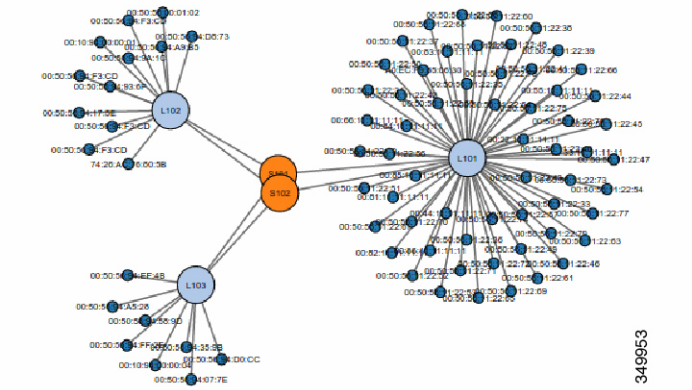


図 *9:* エンドポイントの場所の示力図



ACI Lint

コンピュータプログラミングでは、リントは不一致の識別、または一般的なエラーの単純なデバッグツールを参照する用語です。ACIがインフラストラクチャをコードとして提供するという意味では、ACI がリント アプリケーションを備えているのは妥当と言えます。ACI ツールキットにはそれが備わっています。ACI Lint は、次の 2 つの主要機能の構成エラーを確認し、オペレータに通知するアプリケーションです。

セキュリティの問題：セキュアかセキュアでないかのいずれかとして EPG にタグを設定する機能をサポートし、セキュリティ境界を越えてコントラクトが使用されていないことを検証します。

構成の問題：共通する構成エラーを確認し、ユーザに報告します。

参考のため、出力例を以下に示します。

```
user@linuxhost:~/acitoolkit/applications/lint$ ./acilint.py
Getting configuration from APIC....
Processing configuration....
Critical 001: EPG 'default' in tenant 'infra' app 'access' is not assigned security
clearance
Critical 001: EPG 'x' in tenant 'common' app 'default' is not assigned security
clearance
Warning 001: Tenant 'Cisco' has no Application Profile.
Warning 001: Tenant 'Books' has no Application Profile.
Warning 001: Tenant '3tierapp' has no Application Profile.
Warning 001: Tenant 'mgmt' has no Application Profile.
Warning 002: Tenant 'Books' has no Context.
Warning 002: Tenant '3tierapp' has no Context.
Warning 004: Context 'oob' in Tenant 'mgmt' has no BridgeDomains.
Warning 005: BridgeDomain 'CiscoBd' in Tenant 'Cisco' has no EPGs.
Warning 005: BridgeDomain 'inb' in Tenant 'mgmt' has no EPGs.
Warning 006: Contract 'default' in Tenant 'common' is not provided at all.
Warning 006: Contract 'WebServers' in Tenant 'Acme' is not provided at all.
Warning 006: Contract 'External' in Tenant 'Acme' is not provided at all.
Warning 007: Contract 'default' in Tenant 'common' is not consumed at all.
Warning 007: Contract 'WebServers' in Tenant 'Acme' is not consumed at all.
Warning 007: Contract 'External' in Tenant 'Acme' is not consumed at all.
Warning 007: Contract 'outside-to-web' in Tenant 'roberbur' is not consumed at all.
```

ACI ケーブル プラン モジュール

データ センターのサポートにおいて、ケーブルの管理はきわめて重要です。ケーブル配線に問題があると、データ センターで何かを新たに展開するときに数時間の遅延が起きる場合があります。Cable Planモジュールを使用すると、プログラマは、XMLファイルから既存のケーブル プランを容易にインポートしたり、Application Policy Infrastructure Controller (APIC) コントローラから現在実行中のケーブル プランをインポートできます。また、以前にインポートしたケーブル プランの特定ファイルへのエクスポート、ケーブル プランの比較もできます。上級ユーザであれば、Cable Planを使用して、ケーブル プランのXMLファイルを容易に作成できます。また、ケーブル プランの問い合わせ、ケーブル プランの変更もできます。

ACI CLI エミュレータ モジュール

Cisco Application Centric Infrastructure (ACI) には新しい概念が多数採用されているため、最初のうちはそのポリシー主導型アーキテクチャを理解できないユーザもいるかもしれません。このポリシー主導型アーキテクチャと使い慣れたCLIとの統合を望むユーザには、ACI CLIエミュレータがその作業を支援します。CLIエミュレータ モジュールを使用すると、ACIに固有のテナント、コンテキスト、および他のポリシーに対し、**show** コマンドを実行できます。

ACI マルチサイト アプリケーション

マルチサイト アプリケーションを使用すると、Cisco Application Centric Infrastructure (ACI) ファブリックを複数のサイトにまたがって拡張できます。各サイトは、ファブリックがそれぞれ独自の Application Policy Infrastructure Controller (APIC) クラスタを持つ独立した ACI ファブリックになります。マルチサイト アプリケーションは、コントラクトを複数サイトに拡張させることで、 ACI のグループ ベースのポリシー モデルを保持します。これにより、異なるサイトのエンドポイント グループ間の通信が可能になります。

ACI 設定のスナップショットおよびロールバック

Snapback は、Cisco Application Centric Infrastructure (ACI) ファブリックの設定をスナップショットおよびロールバックするためのツールです。具体的には、管理者はこのツールを使用して次のタスクを実行できます。

- 実行中の ACI ファブリックの設定のライブ スナップショット
- ワンタイム スナップショットと繰り返しスナップショット（即時およびスケジュール済みの両方）
- 設定のバージョン付けしたストレージ
- スナップショット間の違いを含むスナップショット設定の完全表示
- 以前の設定スナップショットへの完全または部分的ロールバック
- Web ベースまたはコマンドラインによる管理

ACI EventsFeeds：ACIアトム フィードへのイベント

EventFeeds アプリケーションでは、オブジェクトが更新されている場合に、WebSocket 接続での Application Policy Infrastructure Controller (APIC) 管理対象オブジェクトへの登録と記録を行います。これらの更新は、HTTP で提供されるさまざまな Atom フィードに表示されます。

Atom フィード アプリケーションに対するCisco Application Centric Infrastructure (ACI) イベントの使用例として、次のものがあります。

- NOC での最近のエンドポイントの表示
- IPTV での更新されたテナントの表示
- フィード クライアントでのエンドポイント グループの変更のモニタリング

GitHub

ソース管理

オープン ソース ソフトウェアはIT で一般的な動きであり、コンシューマ ソフトウェアやWeb サーバ、データベース、ならびにオペレーティング システム全体でさえも含む、多くのプロジェクトの成功の背後にある動機になっています。オープン ソースの成功の重要な側面の1つに、世界中の多くの開発者が1つのプロジェクトで一緒にコラボレーションする機能があります。Concurrent Version Control（CVS）や Subversion（SVN）のような以前のツールは、ソースコードの共通データベースを維持する中央管理サーバを使用して多くの開発者が一緒に作業できるようにするために使用されていました。これらのツールが十分に機能し続ける一方で、サーバベースのツールから集中管理されていないユーティリティへの移行は遅々としていました。Git は一般的なオープンソース オペレーティング システム Linux の作者である Linus Torvalds によって作成されました。Git には、完全なローカル リポジトリのコピー、分散アーキテクチャ、効率に優れたブランチのサポートなど、その他のほとんどのソース管理ツールよりも優れた点がいくつもあります。

GitHub

GitHubは、Gitに基づいたホスティングプラットフォームで、無償および有償のホスティングサービスを提供しており、個人が800万人を超える他のGitHubユーザとプロジェクトで一緒にコラボレーションできるようにします。Git関連のラッパーとは別に、GitHubは問題のトラッキング、プロジェクトへのアクセスの保護、組み込み型のプロジェクトのマニュアル化を行うテクニックも提供します。これらすべての機能を組み合わせることで、GitHubはコミュニティメンバーの共通の場所となり、互いにコードを共有し、互いの作業を足掛かりとし、自分たちの取り組みをより大規模なプロジェクトに貢献させています。

GitHubには、特定の言語に限らず、通常は、ソースコードを保存しますが、Gitプロトコル自体があらゆるファイルタイプのストレージやバージョン管理をサポートしているため、ユーザがマニュアルやその他の常に変更されるファイルをGitに保存することは珍しくありません。主な利点は、Gitが提供するバージョン管理によってユーザはファイルを以前保存したバージョンに戻したり、新しいバージョンに戻したりできることにあります。Gitには、ファイルに加えた変更の監査も維持され、ファイルのバージョンを分岐させる高度なサポートも備えていることから、複数の同時変更をファイルに行うことができ、作業完了後にそれらをマージすることができます。

「It's on github」

「It’s on github（それはGitHubにある）」は、IT業界で良く使用されている最新の用語で、GitHubを使い慣れているユーザにとっては、ダウンロード、変更、プロジェクトへの貢献を誘う言葉ですが、経験のないユーザにとっては複雑なトピックのように思えます。実際には、GitHubは非常に簡単に使用できるツールです。GitHubの保存されている情報は、プロジェクトのメイン ページにアクセスし、プロジェクトのメイン ページの右下にある [Download ZIP] を見つけるだけで利用できるようになります。この方法でダウンロードしたファイルには、プロジェクト内のファイルの最新のバージョンが含まれています。ユーザがこのファイルで何を行うかは、その内容によって大きく異なりますが、GitHubで最も勧められている行為の1つは、プロジェクトに関する明確で明快なマニュアルを提供することです。それにより、新しいユーザがGitにあるプロジェクトのフロントページにアクセスしたときに最初に目にするページで、そのプロジェクトのダウンロード方法とインストール方法を見つけることができます。

プロジェクトへの貢献を考えているユーザの次のステップは、GitHubのアカウントに登録し、グラフィック ベースのクライアントをダウンロードしてコマンドライン ベースのGit ツールへのシンプルなインターフェイスを提供することです。GitHub自体にはグラフィカル クライアントが用意されています。Windowsバージョンは http://windows.github.com で、Macバージョンは http://mac.github.com で入手できます。その他の一般的なソース管理ツールには、SourceTree from Atlassian（http://sourcetreeapp.com で入手可能）などがあります。

アカウントとGitHubクライアントを取得すると、ユーザは、使用可能なプロジェクトを自分のプライベート リポジトリに「分岐」または分割し、変更を加えて、プライベート ブランチにコミットして戻します。それらの変更が機能し、ユーザが元のプロジェクトへの寄与を希望する場合は、「プル」要求を送信できます。原則的には、ユーザは自分の作業を元のプロジェクトにプルバックする必要があることを提示していることになります。プロセスはこのように単純ですが、多くのより高度なプロジェクトには、それらに貢献するための基準やルールがあり、作業をプロジェクトにコミットバックする方法が整備されているため、貢献を試行する前に目を通しておく必要があるかもしれません。