自分のコンピュータを設定する方法
自分のマシンでこのラボの作業を行うには、PuTTY または Terminal などのターミナル エミュレータが必要です。

このラボ演習には Python インタープリタと Spark アプリケーションが必要になります。

このラボの作業を行うには、事前設定済みの dCloud ラボを使用するか、自分のコンピュータを設定する必要があります。詳細については、イベント前の準備モジュールを参照してください。

ミッション：Python を使用して Spark にメッセージを投稿する

おめでとうございます。最初のミッションに着手するところまでたどりつきました。このラーニング ラボの目的は、ラボ環境の最終設定を検証するとともに、必要なラボ トポロジのプラットフォームに接続して DevNet Express ラボを一通り実施できることを確認することです。ここでは、dCloud ラボの APIC-EM および CSR 1000V とのインターフェイスとして機能する Python コードを作成することによって、このタスクを実行します。

まず、Spark と呼ばれるシスコのメッセージング プラットフォームの紹介から始めます。

インスタント メッセージは、チームメイトとの対話ほか、ヒューマン マシン インタラクション（人と機械の間の相互連携）にも利用できる優れた方式です。Cisco Spark は、グループ チャット、コンテンツ共有、デスクトップ共有のためのプラットフォームです。また、Cisco Spark は、REST API、エンドツーエンドの暗号化、複数のプラットフォームに対応したクライアントアプリケーションを提供します。これにより、Cisco Spark は、運用ワークフローおよび DevOps ワークフローのための有望な選択肢となっています。

ここでは、Spark の紹介を通じて、開発者がメッセージング ツールを開発プロセスの一部として活用する様子を紹介します。たとえば、ネットワークがダウンしたときに自動ボット（Bot）でテキストを受信できたらどうでしょう。または、ボット（Bot）の使用によって、自動化された設定変更を導入したり、自分やチームがアラートを受け取ったりできるのはどうでしょうか。便利だと思いませんか? ラップトップを開けたり PuTTY セッションを開始したりする必要もないのです。

開発者は、このタイプのタスクを日常的に実行します。メッセージ アプリケーションを使用して、実稼働への導入を許可したり、新しいコードがユニット テストや統合テストを通らなかった場合にツールからアラートを受け取ったりします。これらのアラートは、電話、タブレット、メッセージング クライアントから送信および応答することができます。

その動作の様子を説明するために、ここでは Cisco Spark プラットフォームとのインターフェイスとして Spark API を使用し、いくつかのサンプル メッセージを送信します。インストラクタ指導のセッションに参加している場合は、講座を通じて、みんなで共通の Spark ルームを使用します。ラボを独習する場合でも、Spark ルームにメッセージを投稿することで任意のシステムや機器と連携する様子を確認することができます。Spark については REST API および Python モジュールでもさらに詳しく説明していますので、続けて詳細をご確認ください。

目標

所要時間：30 分

- Spark REST API を使用して、Spark ルームを名前で検索し、必要な場合は新たに作成する。
- Python を使用して Spark ルームにメッセージを投稿する。
- スクリプトを使用して、ラボのデバイスに関するメッセージを Spark ルームに投稿する。

前提条件

- このモジュールを実行するには、イベント前の準備モジュールを完了している必要があります。
- 必要なアプリケーション、開発環境、およびツールをインストールするための手順は、イベント前の準備モジュールで提供しています。

背景

- Python に馴染みのない方は、REST API および Python モジュールをご確認ください。ここでは、エディタを利用して一部の変数を変更することで、ラボ内のデバイスと Spark を連携させることができます。

Git Repo を複製する

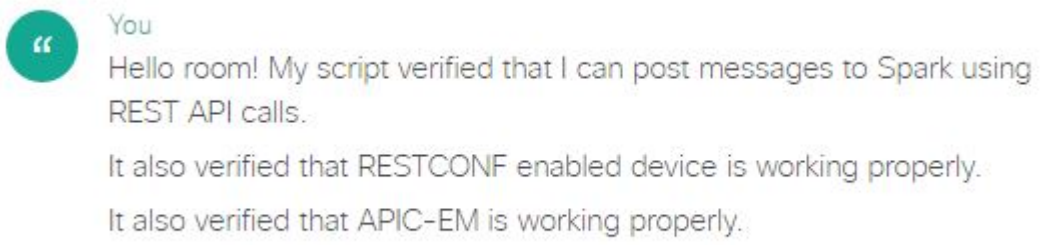
- DevNet Express のコード例を複製します。
    - この GitHub リポジトリでは、このラーニング ラボで実行できるサンプル コードが提供されています。
- 詳細な手順については、このモジュールの以前のラーニング ラボを参照してください。

このミッションを行うには 2 つの方法があります。1 つは、すでに GitHub リポジトリにあるものを再利用する方法（こちらのほうがシンプルです）で、もう 1 つは、独自のスクリプトを作成する方法です。独自のコードを作成すると、Spark API をより深く知ることができます。以前のラーニング ラボで複製した GitHub リポジトリには、`hello_lab.py` と `myspark.py` の 2 つのファイルがあります。独自のコードを最初から作成するのを省きたい場合は、これらのファイルを利用できます。

ステップ 1：Spark ルームを作成してメッセージを投稿する

ミッションを始めましょう。最初のミッションとして、Spark チャット ルームにメッセージをいくつか投稿したいと思います。これには 2 つの理由があります。1 つめは、インストラクタ指導のセッションでこれらの演習を行っている場合に他の受講者に作業の成果を披露したいからです。2 つめの理由は、導入部で述べたように、自動化ソリューションをサードパーティ アプリケーションと統合できることを強調するためです。たとえば、ネットワーキング デバイスでいくつかの設定変更を自動化したい場合を考えてみましょう。変更が発生したときに自動化ツールがアラートを送信できたら便利だと思いませんか。手始めに、カスタム アプリケーションに Spark を統合する方法を説明します。

独自のコードを最初から作成する場合は、Python を使用してスクリプトを作成し、Ubuntu ホストまたはローカル マシンからそのスクリプトを実行することで、APIC-EM コントローラの REST API および RESTCONF/NETCONF を実行する CSR1000V に接続できることを確認することがゴールになります。コードの結果は Spark ルームに投稿されます。コードの結果は次のように表示されます。



GitHub リポジトリの既存のコードを使用する場合は、次の手順に従います。

まず Python を使用して Spark ルームにいくつかの簡単なメッセージを投稿することから始めましょう。必要に応じて、ローカル ホストまたは dCloud ラボの Ubuntu ホストでターミナルを開き、適切なディレクトリに移動します。

```
$ cd devnet-express-code-samples/module03/03-environment-03-mission/
$ ls
hello_lab.py myspark.py
```

ここで、好きなエディタを使用して、`hello_lab.py` ファイルを更新します。このコードには、Spark と通信するための関数と名前がいくつか含まれています。あるいは、理解できたらロジックを最初から作成することもできます。

Spark API トークン

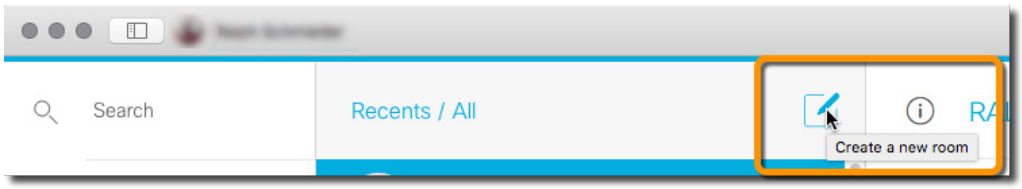
`SPARK_TOKEN` の値には自分専用の Spark API トークンを使用する必要があります。13 行目に移動して、文字列を、以前のラーニング ラボで取得したトークンに変更します。必要な場合は、以前のラーニング ラボを再度確認してこのタスクの実行方法を復習してください。

```
# トークンおよびその他の情報を指定する必要があります。
# 独自のルームを作成した場合はその名前を、
# DevNet Express イベントで共通のルームを利用する場合は、共通のルーム名を指定してください。
SPARK_TOKEN = 'developer.ciscospark.com から取得したトークンをここに挿入します'
```

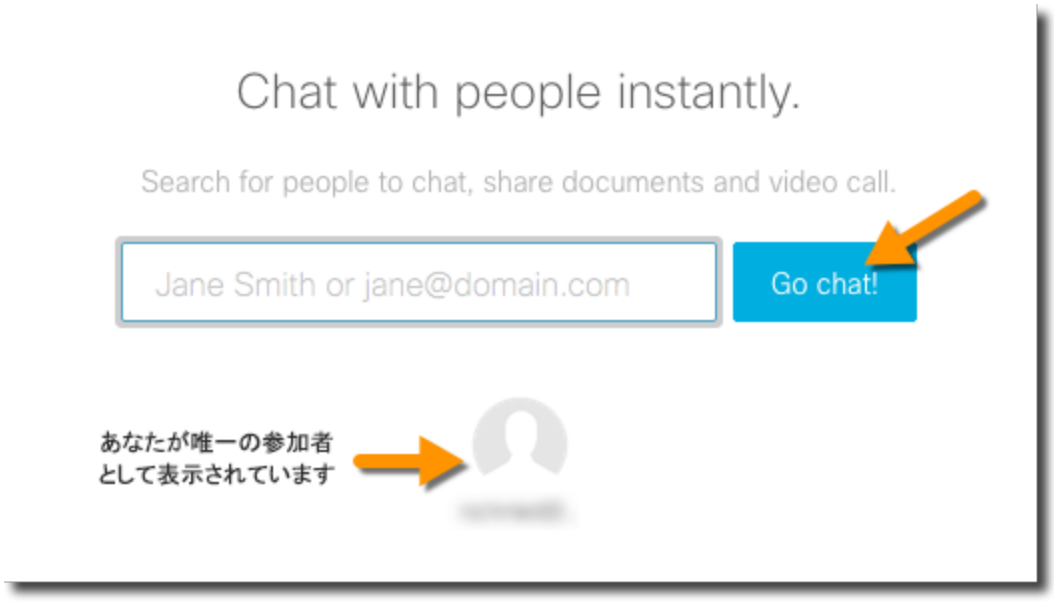
Spark ルーム名

`SPARK_ROOM` の値には、メッセージを投稿したいルームの名前を入力する必要があります。

- インストラクタ指導のコースに参加している方は、投稿先のルーム名をインストラクタに聞いてください。
- このラボを独学で実行している場合は、Cisco Spark アプリケーションを使用してルームを作成する必要があります。以下に、アプリケーションを使用してルームを作成する手順を示します。
    - Spark クライアントを開始します。
    - 新規ルームを作成します。



    - 電子メール アドレスを追加し、Enter を押します。
    - [チャットを始める（Go chat!）] をクリックします。



    ルームは、メッセージを送信しないと開始されません。

    - メッセージを入力します。ルームは、「Untitled」というタイトルで表示されます。
    - タイトルをクリックし、ルーム名を「MY FIRST ROOM」などの覚えやすい名前に変更します。
- 14 行目に移動して、文字列をルーム名に変更します。

```
SPARK_ROOM = 'ここには既存のルーム名を挿入します'
```

注：提供された `my_spark` ライブラリ関数を使用してルーム ID を取得したら、`print()` 関数を使用して、ID を画面に出力し、コード内のルーム ID を永久に置き換えることができます。ルーム検索を省略できるとコードの動作が若干速くなります。

RESTCONF の URI、ユーザ名、パスワード

稼動している RESTCONF 対応デバイスの IP アドレスまたは URL 情報を入力する必要があります。その情報は、変数 `RESTCONF_URL` で保持します。18 行目に移動して、文字列を、RESTCONF 対応デバイスの IP アドレスまたは URL に変更します。

```
RESTCONF_URL = '198.18.133.218:8008'
RC_USER = 'admin'
RC_PASS = 'C1sco12345'
```

注：dCloud 提供ラボを使用している場合は、RESTCONF の変数にはすでに正しい URL、ユーザ名、パスワードが設定されています。何も変更する必要はありません。

また、RESTCONF サービスにアクセスするには正しいポートを入力する必要があることにも注意してください（ここでは 8008）。

APIC-EM の URI、ユーザ名、パスワード

最後に、稼動している APIC-EM サーバの IP アドレスまたは URL 情報を入力する必要があります。その情報は、変数 `APIC_EM_URL` で保持されます。10 行目に移動し、文字列を、APIC-EM の IP アドレスまたは URL に変更します。

```
APIC_EM_URL = '198.18.129.100'
AP_USER = 'admin'
AP_PASS = 'C1sco12345'
```

注：dCloud 提供ラボを使用している場合は、APIC-EM の変数にはすでに正しい URL、ユーザ名、パスワードが設定されています。何も変更する必要はありません。

また、APIC-EM サービスにアクセスするには正しいポートを入力する必要があることにも注意してください（ここでは HTTPS が使用されているので暗黙的に 443 となります）。

次に進みましょう。これで、当面必要なすべての変更が完了しました。次のラボ演習の手順では、ラボ トポロジで必要なすべてのデバイスに接続できることや、Spark ルームに投稿できることを確認します。

自分のコンピュータを設定する方法
自分のマシンでこのラボの作業を行うには、PuTTY または Terminal などのターミナル エミュレータが必要です。

このラボ演習には Python インタープリタと Spark アプリケーションが必要になります。

このラボの作業を行うには、事前設定済みの dCloud ラボを使用するか、自分のコンピュータを設定する必要があります。詳細については、イベント前の準備モジュールを参照してください。

ステップ 2:結果の表示

このラボ手順では、`hello_lab.py` の例を実行して、今後のラーニング ラボで使用するラボ環境を検証する方法を示します。この Python スクリプトは、Spark ルームにメッセージを投稿するために必要な関数を保持する `myspark.py` の名前空間をインポートします。

開始するには、Spark アプリケーションを開きます。必要に応じて、「環境の設定」モジュールの設定に基づくクレデンシャルを使用して認証します。

次に、ターミナルを開き、`hello_lab.py` コードがあるディレクトリに移動します。

```
$ cd devnet-express-code-samples/module03/03-environment-03-mission/
$ ls
hello_lab.py myspark.py
```

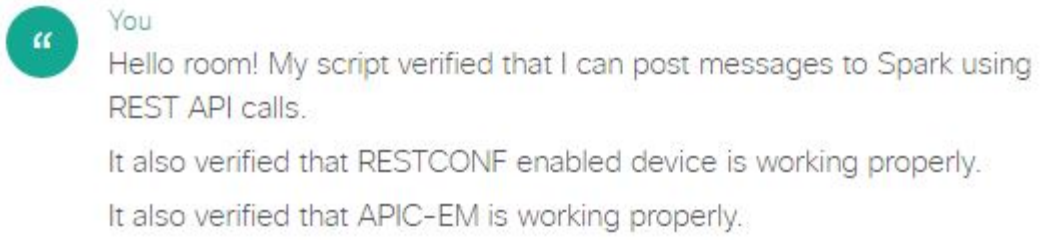
注:`myspark.pyc` も表示されることがあります。これは Python バイト コードです。つまり、Python ライブラリが他のスクリプトによって使用されている場合(ここでは `myspark.py` が `hello_lab.py` によってインポートされています)、Python がスクリプトをバイト コードに*コンパイル*し、その結果を `.pyc` ファイルに保存することで、その後同じライブラリを素早く使用できるようにしています。
`hello_lab.py` スクリプトを実行して、結果を見てみましょう。

```
(mycode) $ python3 hello_lab.py
Your room ID "AABBCCDDEEFF-SOME-LONG-STRING-EEFFAADD".
Please check room YourTestRoom, there are messages posted on your behalf.
(mycode) $
```

注:コマンド ライン プロンプトの前の括弧は、Python Virtual Environment(Python の仮想環境)の「(mycode)」が起動中であることを示します。Python 環境を自分のマシンにインストールして 、かつ、Virtual Environment(Python の仮想環境)を使用している場合は、適切な Virtual Environment(Python の仮想環境)が起動中であることを確認してスクリプトを実行してください。
そうしないと、必要なライブラリが見つからず、実行に失敗する可能性があります。

ただし、すべてのライブラリをグローバルな Python 環境にインストールしてある場合は、その必要はありません。詳細は、「ラボの準備」モジュールまたは「次に進む前に」モジュールを参照してください。

では、Spark アプリケーションについて見ていきましょう。自分のスクリプトから新しいメッセージがルームに投稿されているはずです。



背景情報として、何が起きたのか説明すると、`hello_lab.py` スクリプトが REST API コールを使用してラボ内のさまざまなコンポーネントに実際に接続しています。次に、再び REST API コールを使用して、テストしたデバイスの到達可能性に関するいくつかのステータス メッセージを所定の Spark ルームに投稿しています。

ラボの機器に接続できることを確認してみましょう。また、Python の大量の関連ライブラリが揃っていることも確認しましょう。このスクリプトは、ラボの CSR1000V 上の APIC-EM コントローラおよび RESTCONF DMI エージェントに API コールを行いました。上記の出力に基づき、これらの操作が成功したので、Spark ルームに適切なメッセージを投稿しました。

自分の結果が上記と異なっていたら、このラーニング ラボを注意深く復習して、すべての手順が実施されたか確認してください。また、ラボ環境モジュール内のその他のラーニング ラボや「環境の設定」モジュールを参照し、すべての前提条件が整っているか確認してください。

おめでとうございます。さらに先に進み、次のラーニング ラボで取り組みを続けましょう。