Setting Up the Spark Lab Environment

The purpose of this lab is to provide an additional set of steps for lab setup and validation. This learning lab contains information necessary to ensure users may post messages to Spark using Python in the next learning lab.

Note that users who are not familiar with Spark shall be introduced to the required topics in the next learning lab and the REST APIs and Python module. The purpose of this learning lab is to introduce some of the basic setup.

Objective

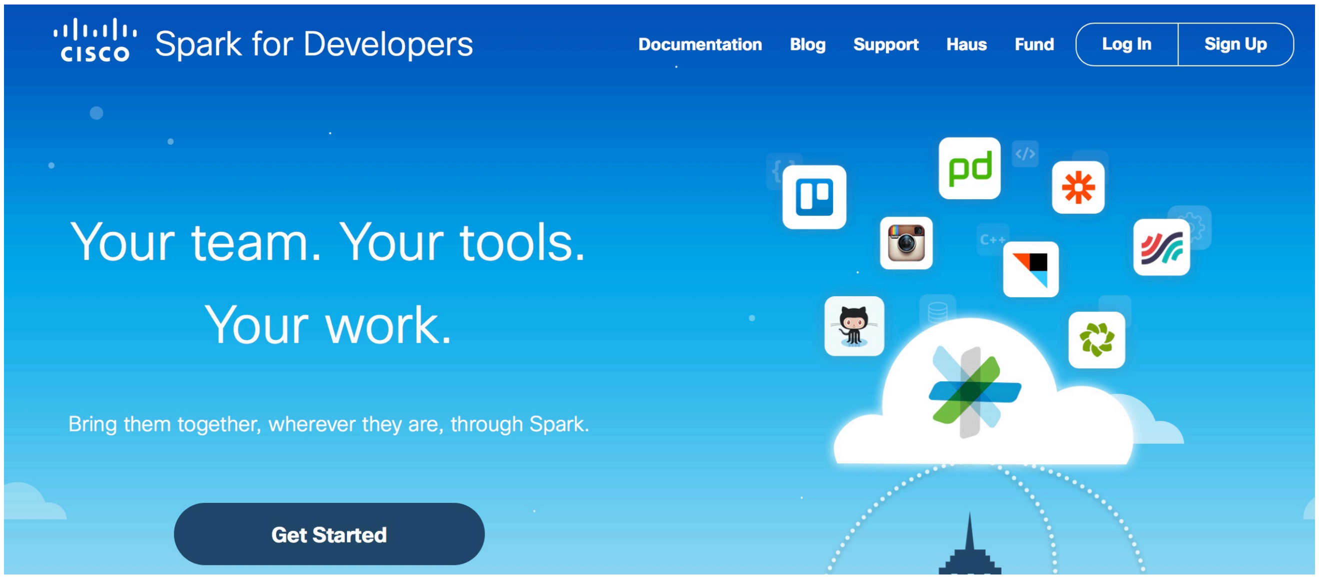
Completion Time: 15 minutes

- **Navigate to the Spark Developer Portal**
- **Retrieve your Spark API Token**

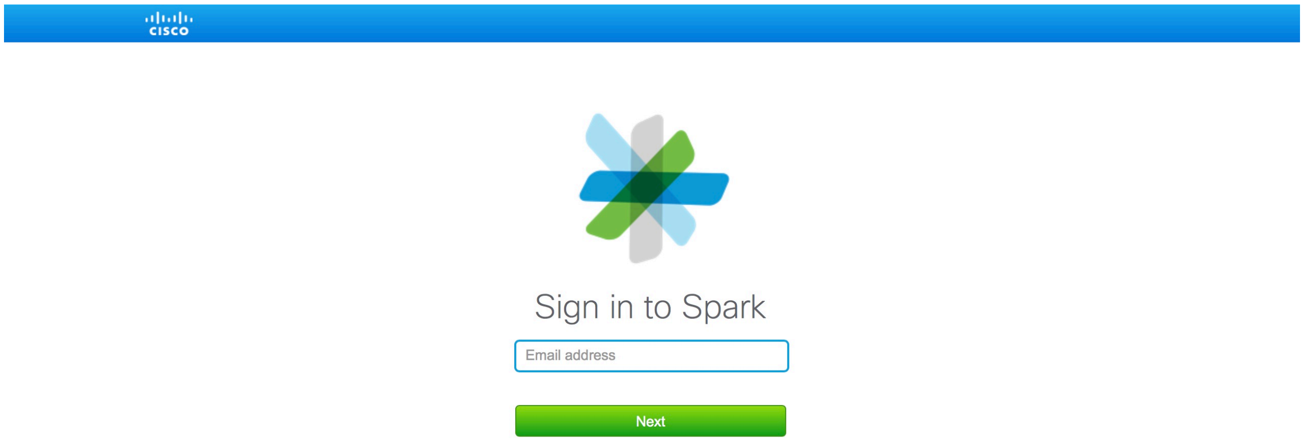
Step 1. Retrieving Your Spark Authentication Token

First, we have to retrieve our Spark developer API token. We use this token to authenticate API calls with the Spark platform.

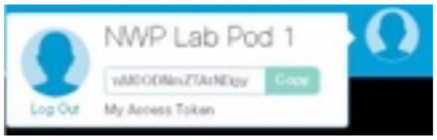
To do so, use a browser to go to: https://developer.ciscospark.com. This is the URL to access the Spark developer page. We first have to retrieve our authentication token to complete subsequent labs.



Click `Log In` in the upper right and use your Spark credentials to authenticate. If you do not have Spark credentials, refer to any setup steps in the Pre-Event Preparation module.



After authenticating using the browser, click on the icon in the upper right. You should see your token.



Click the `Copy` button to save this token string. Do not share this information! This enables you to authenticate against the Spark API to post messages and make other changes to Spark rooms. Ideally, save this to a safe file for the next learning lab. Note that tokens are valid for 14 days. After that, you will need a new

token to authenticate.
Continue on to the next lab step to finish this lab setup.

Step 2. Downloading Source Code From the Git Repository

A Word About Git

Git is a distributed software version control system that allows developers to manage the software development lifecycle. GitHub provides git repositories as a service allowing developers of all companies to collaborate on projects.

The information below describe how to download the most recent sample code, so that you may complete all learning labs.

Getting the Source Code Files

Many of the DevNet learning labs have source code that must be downloaded and run to complete the labs. If you do not have the repository source code on your workstation you will need to copy the source code from the Git repository `Federal_DevNet_Express_Q3KSO.git`. Git uses the **clone** command to copy files. To accomplish this task see the section **Cloning the Git Repository** below.

If you already have the source code on your workstation or are working in the DevNet provided dCloud environment which already contains the source code files, you will need to update these files. Git uses the **pull** command to update files. To complete this step see the section **Pulling the Latest Files From Git** below.

Cloning the Git Repository

Since you've determined that your workstation does not have the source code files, let's clone the source code for the labs. Open your favorite terminal like `cmd.exe` or PowerShell on Windows, Terminal.app on Mac OS, or XTerm on Ubuntu ...etc.

It is suggested that you create a dedicated directory to clone the sample code as outlined below as well as to add your own scripts and files. Change into that directory and then perform the steps below.

```
$ mkdir Code
$ cd Code
$ git clone https://github.com/marknguy/Federal_DevNet_Express_Q3KSO
```

```
.
.
$ cd Federal_DevNet_Express_Q3KSO/module03
.
.
$ ls
$ hello_lab.py myspark.py
```

Pulling the Latest Files From Git

If you've previously cloned the source code files to your workstation or if you are using the DevNet provided dCloud environment which already contains the source code files you should still update your directory with the latest files. Please note that in the DevNet dCloud environment that the `Code` directory already exists in the user's home directory and contains the previously cloned devnet-express-code-samples repository files.

To update the source code files on the Ubuntu host in the dCloud lab or your local workstation open your favorite terminal like `cmd.exe` or PowerShell on Windows, Terminal.app on Mac OS, or XTerm on the Ubuntu host ...etc, then follow the steps below.

```
$ cd <directory where your devnet-express-code-samples repository files resid
e>
$ git pull origin master
.
.
$ cd Federal_DevNet_Express_Q3KSO/module03
.
.
$ ls
$ hello_lab.py myspark.py
```

Excellent! You now have the latest source code samples, and you are ready for the first mission which is to use Python to create a Spark room and post messages!

Mission: Posting Messages to Spark Using Python

Cool! You made it to the first mission. The purpose of this learning lab is to validate any final setup of the lab environment

and ensure that you can reach the required platforms in the lab topology to run through the DevNet Express labs. We shall perform this task by creating some Python code that interfaces with an API (In this case, the API for Spark).
We'll start by introducing you to Cisco's messaging platform called Spark.
Instant messaging is a great way to interact with teammates or even for human-machine interaction. Cisco Spark is a platform for group chat, content sharing, and desktop sharing. Cisco Spark also offers REST APIs, end-to-end encryption, and clients across multiple platforms. This makes it a viable choice for operational and DevOps workflows.
We're introducing you to Spark to show you how developers may use messaging tools as part of the development process. Imagine if an automated bot could send you a text when the network was down. Or imagine using a bot to deploy an automated configuration change and alerting you and the team. Wouldn't that be useful! You wouldn't even have to open your laptop or start a Putty session!
Developers perform this type of task routinely. They use messaging applications to approve deployments to production or receive alerts from tools when new code doesn't pass unit or integration tests. These alerts can be sent and acted upon from your phone, tablet, or messaging client.
To see how this works, we are going to use the Spark API to interface with the Cisco Spark platform and send some example messages. Throughout the class, we will continue to use the same Spark room to communicate with the group assuming you are in an instructor led session. If you are performing the lab on

your own, you can still post messages to a Spark room to see how the third party integration works. Note that we shall cover more about Spark in the REST APIs and Python module, so stay tuned for more details!

Objective

Completion Time: 30 minutes

- Use the Spark REST API to find a Spark room by its name or create it if needed
- Post messages to the Spark room using Python
- Use scripts to post messages about devices in the lab to the Spark room

Background

- If you are unfamiliar with Python, we shall catch you up in the REST APIs and Python module! For now, you can get by with just an editor and changing some variables, so we can integrate with the devices in the lab and Spark.

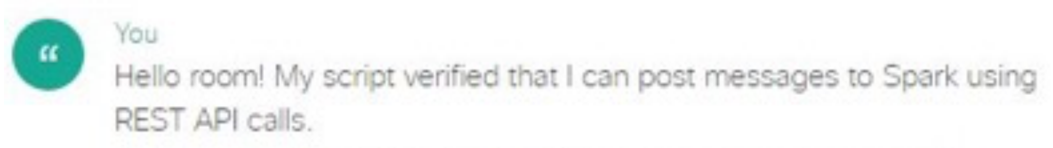
Clone Git Repo

- Clone the DevNet Express example code
    - This GitHub repository provides sample code that you can run for this learning lab
- Reference the earlier learning labs of this module to see the detailed steps

There are two ways for you to approach this mission: either reuse what's already in the GitHub repository (which is simpler) or create your own scripts. If you create your own code, you get the added adventure of exploring the Spark APIs in more detail. The GitHub repository cloned in the earlier learning lab has two files: `hello_lab.py` and `myspark.py.` These files can be used if you do not want to create your own code from scratch.

Step 1. Creating a Spark Room and Posting a Message

Let's start the mission! For our first mission, we want to post a few messages to the Spark chat room. There are two reasons for this. First, for those working on these exercises in an instructor led session, we want to show other students the results of our work! The second reason for this is to highlight how we can integrate automation solutions with third party applications as mentioned in the introduction. For example, imagine we want to automate some configuration changes on networking devices. Wouldn't it be nice if our automation tools would alert us when changes occur? We'll show you how to integrate Spark with your custom application to get started. Note, if you want to challenge yourself by creating your code from scratch, the goal is to use Python to create a script that runs from your local machine to call an API. The results of your code should be posted to the Spark room. The results of your code should appear as follows.



You
Hello room! My script verified that I can post messages to Spark using REST API calls.

For the those that want to use the existing code in the GitHub repository, follow the steps below.
First let's start by posting a few simple messages to a Spark Room using Python. Navigate to the appropriate directory.

```
$ cd module03
$ ls
hello_lab.py myspark.py
```

Now using your favorite editor, let's update the `hello_lab.py` file. This code contains some functions and names that will allow us to interface with Spark. Alternatively, you can follow along and create the logic from scratch.

Spark API Token

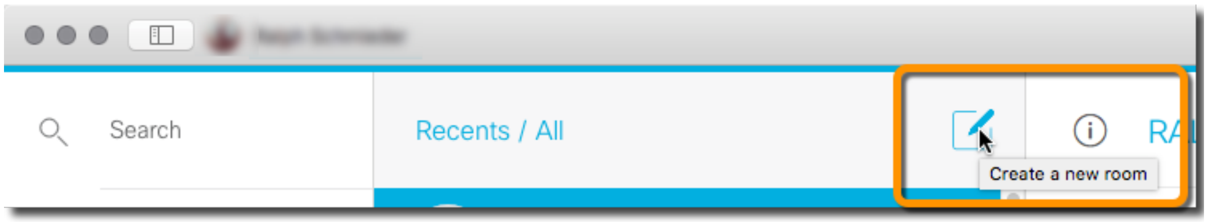
You must use your own Spark API token for the value of `SPARK_TOKEN`. Jump to line 13 and change the string to the token you retrieved in the previous learning lab. Revisit the previous learning lab for a reminder on how to perform this task if needed.

```
# We need to know our token and other information.
# Make sure you either create your own room
# or provide the name of the common event room
SPARK_TOKEN = 'insert-your-token-from-
developer.ciscospark.com-here'
```

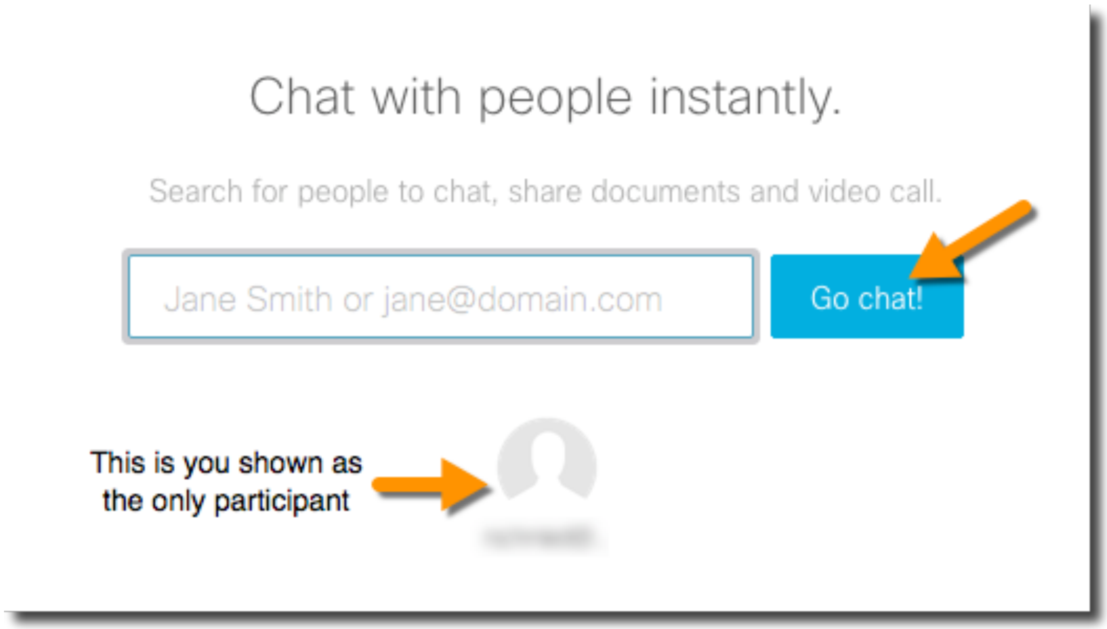
Spark Room Name

You will need to provide a room name for the value of `SPARK_ROOM`, where you would like to post your message.

- **In this case the Spark room has already been filled out for you.**
- If you are doing this lab on your own then you need to create a room using Cisco Spark application. Below are the steps to create a room using the application:
  - Open your Spark client
  - Create a new room



- **Add your email address, press Enter**
- Click 'Go chat!'


Chat with people instantly.
Search for people to chat, share documents and video call.
Jane Smith or jane@domain.com
Go chat!
This is you shown as the only participant

- The room will only be started when you send a message to it.
- **Type a message. The room shows as 'Untitled'.**
- **Click the title and change the room name to something memorable such as 'MY FIRST ROOM'.** Warning: Please use only ASCII characters to avoid any unexpected results.

▪ Jump to line 14 and change the string to the name of a room.

```
SPARK_ROOM = 'existing-room-name-inserted-here'
```

Note: Once you have retrieved your Room ID using the provided `my_spark` library function you can also print that ID using a `print()` statement and then permanently replace the room ID within the code. The code will work a bit faster if the room search is skipped.

Great! That should be all the changes required for now. In the next lab step, let's verify you can reach all of the required devices in the lab topology and that verify that you can post to the Spark room!

Step 2. Viewing the Results

In this next lab step, we shall show you how to run the `hello_lab.py` example to verify your lab environment for the upcoming learning labs. This Python script imports the namespace of `myspark.py` which holds necessary functions to post a message to Spark room.

To start, open your Spark application. If needed, authenticate using your credentials based on the setup from the 'Set up your Environment' module.

Now, open up a terminal and navigate to the directory where the `hello_lab.py` code resides.

```
$ cd module03/03-environment-03-mission/
$ ls
hello_lab.py myspark.py
```

Note: You might also see `myspark.pyc` which is Python byte code. In other words: When Python libraries are used by other scripts (in this case `myspark.py` is imported by `hello_lab.py`) then Python will *compile* the script into byte code and save the result into a `.pyc` file to speed up subsequent use of the same library.

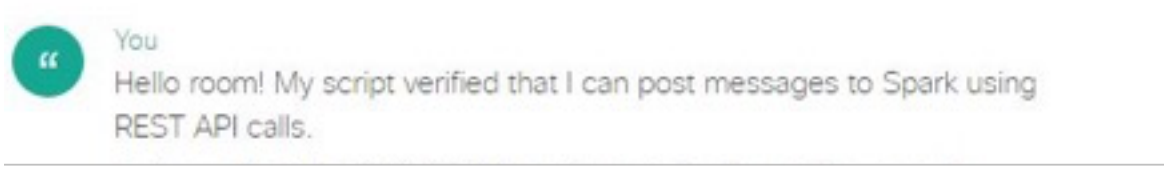
Run the `hello_lab.py` script, and let's see the results!

```
(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) $
```

Note: The parentheses in front of the command line prompt indicate that a Python Virtual Environemnt is active '(mycode)'. If you installed the Python environment on your own machine and you used a Virtual Environment then make sure that it is active when you run this script (or any script, for that matter, that relies on the installation of specific libraries!

This is not needed, however, if you installed all libraries in your global Python environment. See the 'Lab preparation' module or the 'Before you go' module for additional detail.

Now take a look at the Spark application. You should have a new message posted in the room from your script!



You

Hello room! My script verified that I can post messages to Spark using REST API calls.

For some background, here is what just happened: The `hello_lab.py` script actually reaches out to various components inside of our lab using REST API calls. It then prints some status messages about the reachability of the tested devices into the given Spark room, again using REST API calls.

If your results differ from the above, review this learning lab carefully to make sure all steps were performed. Also, take a look at the other learning labs in the `Lab Environment` module and the 'Setup your Environment' module to make sure all prerequisites were covered.

Very cool! Let's charge forward and continue the journey with the next learning lab.