Run a Cisco Spark Bot locally

Duration: 15 minutes

Objectives

As introduced earlier, Cisco Spark Bots are applications that invoke the Cisco Spark API under a Bot account identity.

In this lab, you will learn to run your own Cisco Spark Bot by taking several steps: create a Bot account, run a sample Cisco Spark bot on your local machine, expose your bot to the internet and register Webhooks in order to have Cisco Spark post notifications to your bot. Finally, you will chat with your new friend:

Pre-requisites

You will need a Cisco Spark user account to complete this lab. If you're not a Cisco Spark user yet, click to sign up.

How to setup your own computer

Skip this section if you are using a machine provided by the event organizers.

As we will run a bot sample, this requires a minimal NodeJS development environment on your local machine.

- NodeJS runtime: make sure you have both the "node" and "npm" runtimes installed locally on your machine. This lab has been tested with NodeJS v4.
- [optional] a NodeJS IDE: this lab does not require you to open any NodeJS files locally, but this will be mandatory if you opt for the "to go further" instructions.
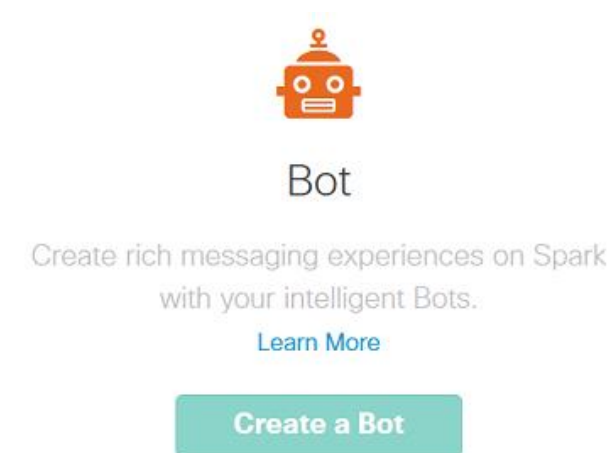- "git" command

As we will visualize JSON payloads, make sure your Web browser embeds a JSON payload beautifier to make this experience more convenient. If you're using Chrome, add json-viewer.

Step 1: Create a Bot account

Log in at "Spark for Developers" and open the "My Apps" menu.
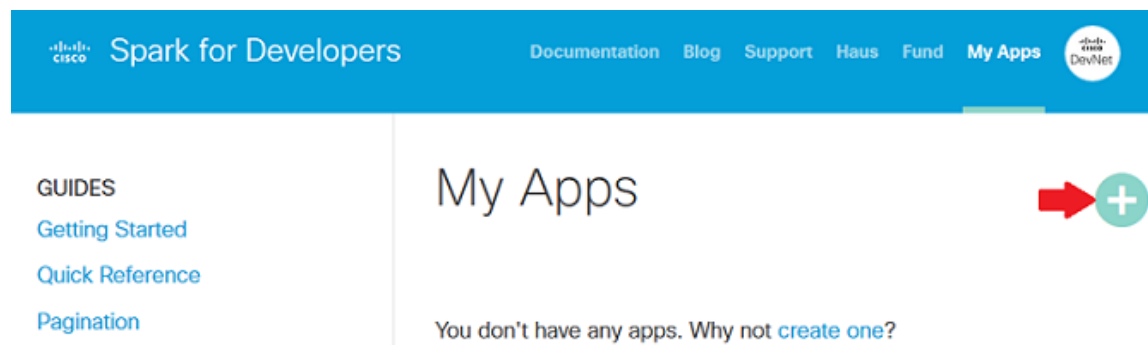
Click on the "+" button, and choose "Create a Bot".

You will access the "New Bot" creation form below.

Fill in a name, a unique email identifier, and specify a publicly accessible image URL with a minimal resolution of 512x512 pixels. Feel free to pick this image example for the sake of this lab.

Note that you will not be authorized to pick the email "my-awesome-bot@sparkbot.io" as it is already reserved. Make sure to replace future mentions to the bot email you have just chosen.

Display Name

The name users will communicate with
after adding your bot to a room

MyAwesomeBot   (bot)

Bot Username

The username users will use to add your
bot to a room. Cannot be changed later.

my-awesome-bot   @sparkbot.io ✓

Icon

A URL to an image that is 512x512 or
larger is required

Remove

Click "Add Bot" to get your Cisco Spark Bot created.

Your Bot's access token is displayed. Paste it in a safe place as it won't show up again, and we will use it in steps 2 and 3 of this lab.

Note that a Cisco Spark Bot access token lasts 100 years. If you ever lose or reveal it, you can come back to this Bot details page and regenerate an access token. The previously issued token will be automatically deprecated.

Access token

ZkMThINjMtYTM1Yi00ZjNjLWI3YTAtMTg5YTkyMDhkOTI3ZmFmZTczM2YtMWQ0   Copy

Make sure to save your access token above. You can always regenerate an Access Token, but
you will not see this one again.

Your bot can now be added to any Cisco Spark Room by specifying its email: my-awesome-bot@sparkbot.io in our example.

Go to your Cisco Spark client, and create a new room with your Bot as a participant.

Search for people to chat, share documents and video call.

my-awesome-bot@sparkbot.io   Go Chat!

Even though you can chat with your bot, you won't see him answer … as we haven't connected it yet to any custom code logic. We'll work on that in the next steps.

Step 2: Run a bot on your local machine

For the sake of this lab, we will clone an existing code sample. This code sample is about a HelloWorld bot that welcomes participants when he's added to a room. The bot then echoes the names of the Cisco Spark users who mention him. We will configure this code sample to run under the Cisco Spark identity of the Bot we just created.

Make sure your local environment meet the git, NodeJS and npm commands pre-requisites.

Open a terminal, and run the commands below to clone the github code sample, install the NodeJS dependencies.

```
> git clone https://github.com/CiscoDevNet/node-sparkbot-samples
> cd node-sparkbot-samples
> npm install
```

We'll now start the bot sample, by running the command below, and pay attention to paste the Cisco Spark token of your own Bot.

```
stsfartz@STSFARTZ-HXDLT MINGW64 /c/Local/node-sparkbot-samples (master)
$ DEBUG=sparkbot*,samples* SPARK_TOKEN=YOUR_TOKEN node examples/helloworld.js
  sparkbot webhook instantiated with default configuration +0ms
  sparkbot addMessagesCreatedListener: listener registered +22ms
  sparkbot:router added listener for command: help +246ms
  sparkbot:router added listener for command: fallback +2ms
  sparkbot:router added listener for command: hello +1ms
  sparkbot addMembershipsCreatedListener: listener registered +0ms
  sparkbot Cisco Spark bot started on port: 8080 +3ms
```

If you're on Mac, Linux, or a Windows Bash shell, run:

```
> DEBUG=sparkbot*,samples* SPARK_TOKEN=YOUR_TOKEN node examples/helloworld.js
```

If you're running the lab in a Windows Command Shell, type:

```
> set DEBUG=sparkbot*,samples*
> set SPARK_TOKEN=REPLACE_WITH_YOUR_BOT_TOKEN
> node examples\helloworld.js
```

**Check your bot is now live by hitting its health check at http://localhost:8080.**

Congrats, you now have your own Cisco Spark Bot up and running!

Step 3: Expose your Bot to the internet

As Cisco Spark is running in the Cloud, it requires your Bot to be reachable via a publicly accessible endpoint. To that endpoint, Cisco Spark will post notifications of what's happening in the rooms your bot is a member of.

For that purpose, we will now create a tunnel between your machine and the internet so that the port your bot is currently listening at, will become publicly accessible.

Here is the architecture we'll setup. This section focusses on the communication labelled 3. in the figure below.
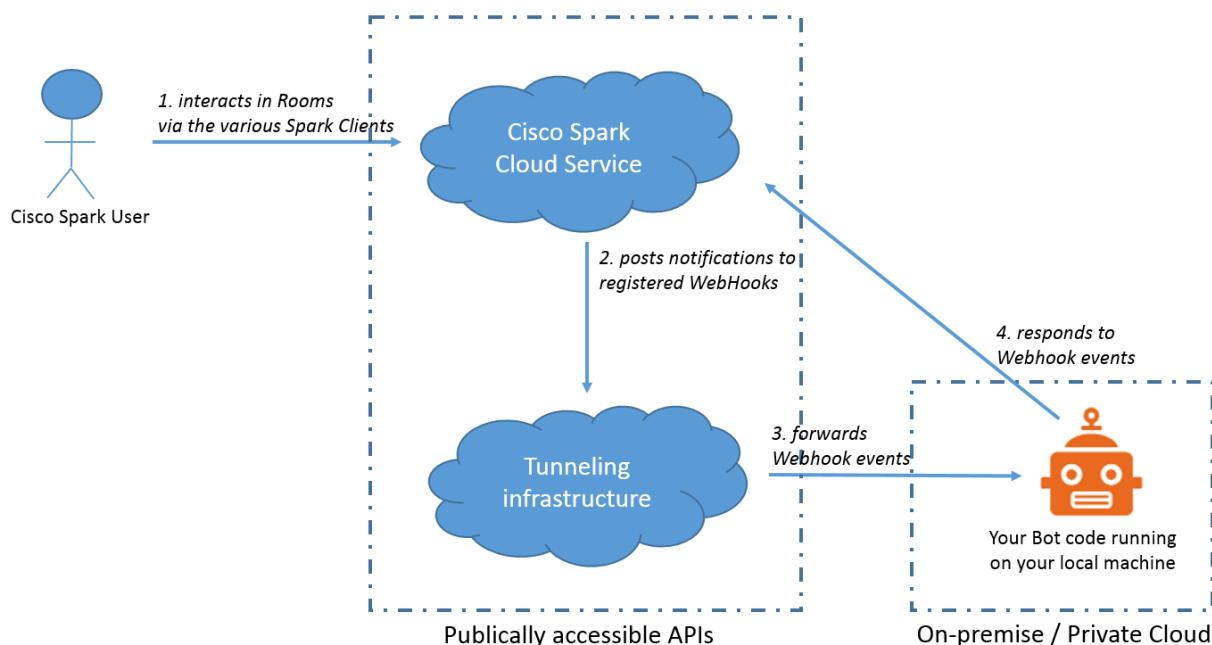
LocalTunnel and ngrok are two popular tunneling technologies, with freemium plans for developers. **This lab leverages the ngrok technology.**

Open a terminal and run "ngrok http 8080".

- *If you're running this lab on your own computer, download the ngrok binary from https://ngrok.com/download.*
- *If you're running this lab at an event, the ngrok executable is either available on the command line, or you'll need to reach to your home directory.*

You'll get an output similar to:

```
Version                 2.1.18
Region                  United States (us)
Web Interface           http://127.0.0.1:4040
Forwarding              http://e86a29c5.ngrok.io -> localhost:8080
Forwarding              https://e86a29c5.ngrok.io -> localhost:8080
```

Your Bot is now exposed live on the internet, and can be reached through HTTP or HTTPS. Look **for the Forwarding lines in the console window of ngrok.** In the snapshot above, you would reach your bot at https://e86a29c5.ngrok.io/

Now open a Web Browser and hit your bot public URL, https://e86a29c5.ngrok.io/ in our case.

Check the health check responds successfully: a JSON payload shows up.

In the next section, we'll dig into more details of this JSON payload.

```
1    // 20161125020422
2    // http://e86a29c5.ngrok.io/
3
4  ▾ {
5      "message": "Congrats, your Cisco Spark bot is up and running",
6      "since": "2016-11-25T01:04:14.545Z",
7      "tip": "Don't forget to create WebHooks to start receiving events from Cisco Spark:
       https://developer.ciscospark.com/endpoint-webhooks-post.html",
8  ▾   "listeners": [
```

Note that the snapshot above leverages the Chrome plugin "JSON viewer".

Step 4: Create Webhook events

In this section, we will create REST Webhooks so that our bot starts receiving notifications from Cisco Spark.

For the sake of this lab, we embed a health-check in our code sample.
In other terms, when someone hits your bot Root URL via the internet tunnel, your bot returns a 200 OK response, with extra configuration and deployment information.

If you look attentively to the JSON data below, you will see your bot is running under the Cisco Spark Identity of the Bot Account you created earlier. Moreover, the "listeners" section lists the events your bot can take action on.

```
{
  "message": "Congrats, your Cisco Spark bot is up and running",
  "since": "2016-10-26T13:00:29.404Z",
  "tip": "Don't forget to create WebHooks to start receiving events from Cisco Spark:
  "listeners": [
    "messages/created",
    "memberships/created"
  ],
  "token": true,
  "account": {
    "type": "machine",
    "nickName": "MyAwesomeBot",
    "person": {
      "id": "Y2lzY29zcGFyazovL3VzL1BFT1BMRS85NjAyZGUzNi1hOWFjLTRiZGUtYmI2OC1hNzAzOTVm
      "emails": [
        "my-awesome-bot@sparkbot.io"
      ],
      "displayName": "MyAwesomeBot (bot)",
      "avatar": "https://c74213ddaf67eb02dabb-04de5163e3f90393a9f7bb6f7f0967f1.ssl.cf
      "created": "2016-10-26T12:45:34.970Z"
    }
  },
  "interpreter": {
    "prefix": "/",
    "trimMention": true,
    "ignoreSelf": true
  },
  "commands": [
    "help",
    "fallback",
    "hello"
  ]
}
```

Our next task is to create the Webhooks.

Open the list of Webhooks supported by Cisco Spark, and look for the event entries needed by your bot:

- Messages / Created: a new message got posted into a room

- Memberships / Created: someone joined a room that you are in

Click on the Webhooks entry in the API Reference section on the left, and select the "Post" method. This will drive you to the Create a Webhook form.

Fill in the fields as shown in the snapshot below:

- Authorization header: change the access token to your bot's, and *do NOT remove the "Bearer " prefix*,
- targetUrl: paste the public URL of your bot exposed by ngrok,
- resource and event: make sure to fill in the "messages" and "created" values, as it is only default placeholders you see in the form,
- last fields "Secret" and "Filter" are optional. Leave them blank.

Then click "Run" and check the response in the right panel.

As your Cisco Spark API call completes successfully, you will see a green "200/success" displayed, and your Webhook will be assigned a unique identifier (check the "id" field). This webhook is fired every time a new message is added to a Cisco Spark room your bot is a member of.

Now, let's create our second Webhook, in order to receive an event every time our bot is added to a room.

On the same form, modify the value of the "resource" field: replace "messages" with "memberships".

Click "Run" again, and check your second Webhook got successfully created, with the "200 / success" message.

Step 5: Test your Cisco Spark Bot

In previous steps, you run a Cisco Spark bot sample, exposed it via ngork, and created Webhooks to receive events. Now comes time to chat with your bot.

Reach to the Cisco Spark room you created in step 1.

Enter /hello.

Check your bot's response!

Mention your bot in Group rooms

As introduced in previous labs, Cisco Spark Bots MUST be mentioned in "Group" rooms in order to be notified.

Open the "Create a Room" form.

Fill in the form with a title. Do NOT modify the token in the Authorization header, so that the room gets created under your personal Cisco Spark identity.

Check the result is a "200/success", and the room type is set to "Group". As you're the only participant in the room for now, Cisco Spark created a "Group" room to let you add participants later.

Now, open your favorite Cisco Spark client, and reach to the newly created room.

Invite your bot to the conversation by adding it as a participant, with the email address you created in step 1.

Type /hello.

You do not get any answer as your bot is not mentioned.

Now mention your bot… and watch for its answer.

To go further

What about extending your bot!

If you have a fully functional development environment at your disposal, we suggest you start by adding a few breakpoints in your bot main file "helloworld.js", and restart your bot in debug mode.

Then, you'll want to add some commands of your own. If you're looking for some inspiration, take a look at these examples.

Note that for the sake of this lab, we leveraged a nodejs bot framework that hides most of the complexity of listening to messages and taking action as commands are fired. Note that several other nodejs Bot frameworks exist for Cisco Spark, from basic to advanced. As an example, Nick Marus's flint framework embeds interesting features such as automated creation of the Webhooks required by your bot… Certainly worth a try…