

Deploy a Cisco Spark Integration

Duration: 15 minutes

Objectives

As introduced in previous labs, Cisco Spark Integrations and Bots concretize Cisco Spark's extensibility.

By creating custom applications, developers can tie together Cisco Spark with existing enterprise processes and data. For example, you may add the [Jira Integration](#) to an existing Room and get the participants notified as Jira issues (EPIC, Story, Task or Bugs) are created or updated.

Cisco Spark Integrations let your applications request permission to invoke the Cisco Spark API on behalf of other users. The process used to request permissions is called "OAuth Grant Flow": it is documented in the [Integrations guide](#), and can be experimented in the Learning Lab [Understand the OAuth Grant flow for Cisco Spark Integrations](#).

In this lab, we'll go through an existing NodeJS example that displays the name of Cisco Spark users. We will deploy it on Heroku, and register it as a Cisco Spark Integration.

How to setup your own computer

This lab can be completed from any platform with an HTML5-compatible browser.

Pre-requisites

In this learning lab, you will deploy an example application on Heroku, and register it as a Cisco Spark Integration.

You will need:

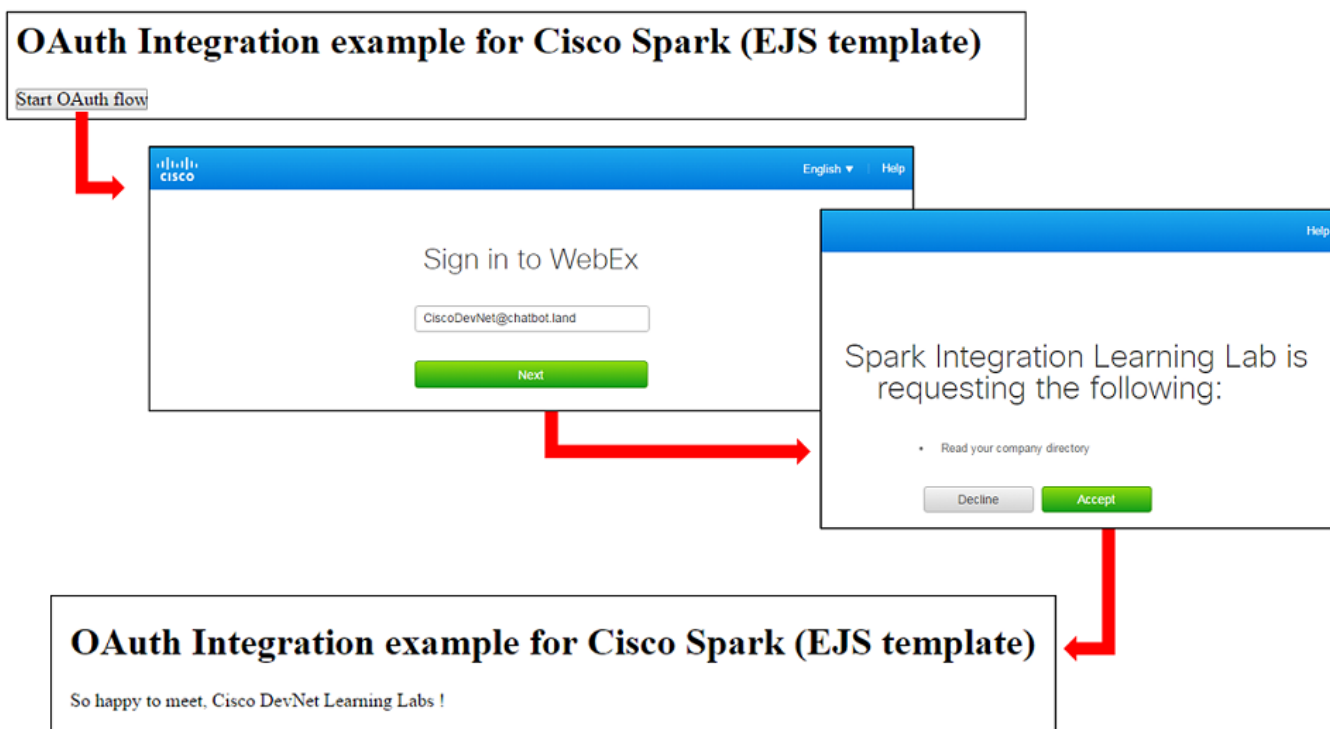
- a Cisco Spark account: if you're not a Cisco Spark User yet, you can [sign up here](#),
- a Github account: [sign up here](#) if you are not already a Github user,
- an Heroku Account: [click here](#) to sign up at Heroku.

Note that for the purpose of this lab, we'll stick to [Heroku free dynos](#) so that the deployment detailed hereafter won't cost you any money.

Step 1: Fork a Cisco Spark Integration

For the sake of this lab, we will leverage a [ready-to-run Cisco Spark Integration](#).

This integration example consists of a [NodeJS server](#) that serves an HTML page to initiate the Spark OAuth Grant flow. This page drives the user to a consent form. The integration also listens on a “redirect URL” that is invoked by Cisco Spark as the end-user accepts or declines to grant permissions.



Open the [Cisco Spark Integration example](#) provided for this lab. Make sure you're signed in on Github.

Click "Fork" to start working on your own copy of this integration.

Step 2: Deploy on Heroku

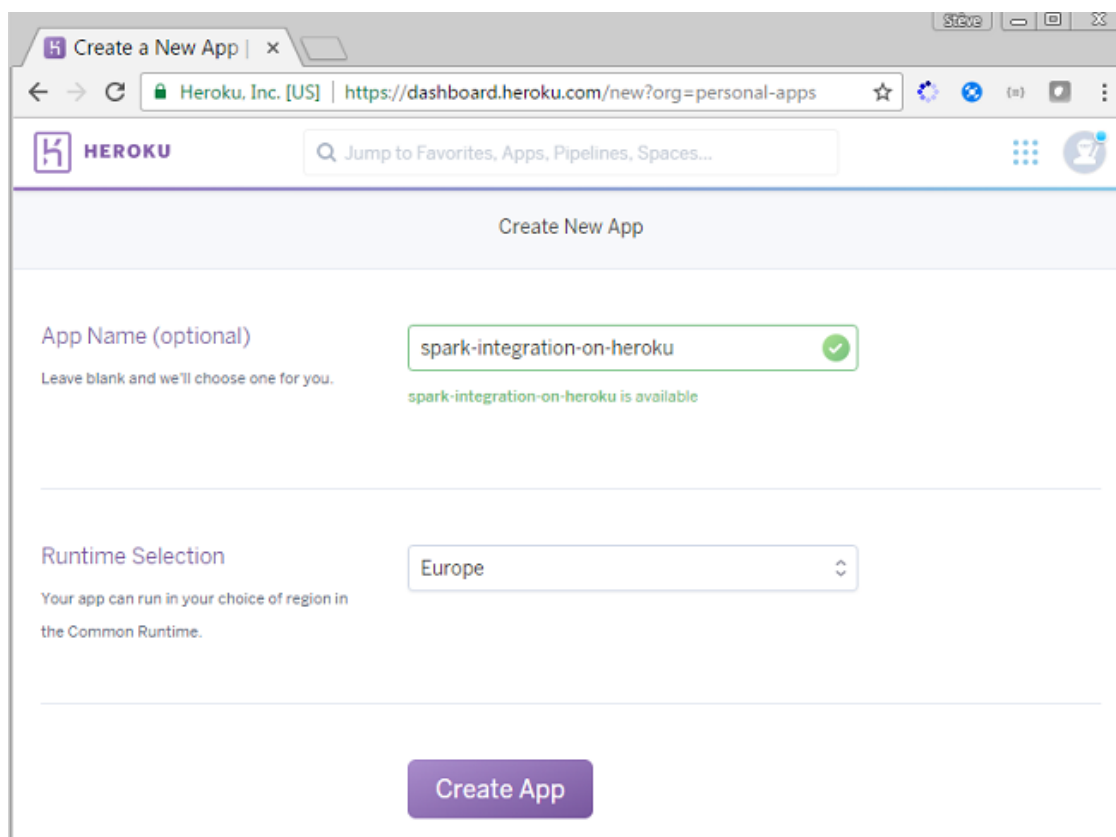
The goal of this step is to create the Heroku project that will host our integration and retrieve its public URL.

Heroku is a Cloud Application Platform that lets you deploy and run code in minutes.

Note that the example provided in this lab can be hosted on other private or public Cloud Application platforms such as Amazon Web Services, Microsoft Azure, Google Cloud, Clever Cloud Enterprise...

*If you don't have a Heroku account yet, you can signup at <https://signup.heroku.com/>. Note that for development and prototyping purpose, we'll stick to Heroku **free dynos** so that the deployment detailed hereafter won't cost you any money.*

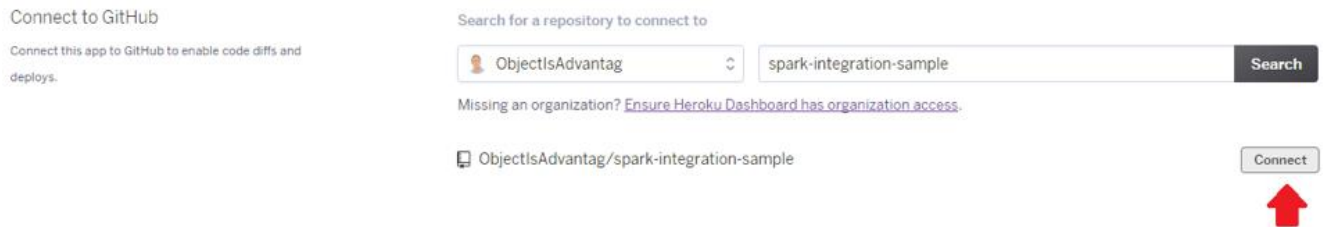
Log on your [Heroku dashboard](#) and create a brand new application.



Once created, Heroku will display your application deployment configuration. The default deployment method is to use Heroku git. For the purpose of this lab, we will deploy straight from Github, and skip the installation of the Heroku CLI (command line interface).

Click on the Github button. Among your Github repositories, select the “spark-integration-sample” project you forked in step 1, and click connect.

If your Github account is not connected to Heroku yet, click on Connect. If you need help, check the [Heroku Github integration guide](#).



Connect to GitHub

Connect this app to GitHub to enable code diffs and deploys.

Search for a repository to connect to

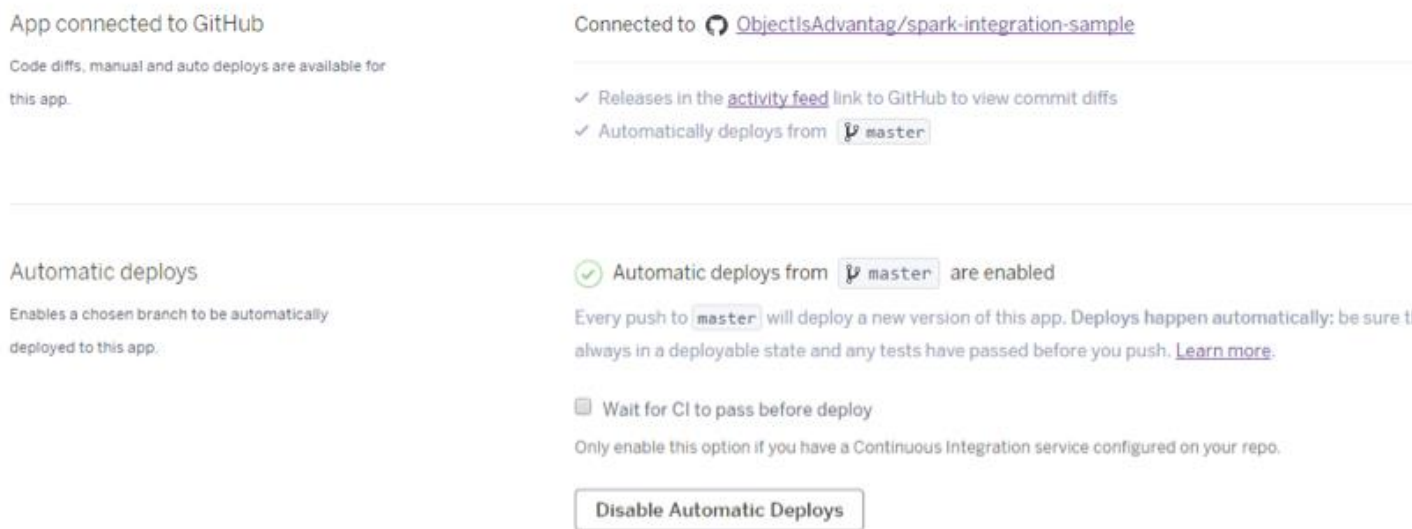
ObjectIsAdvantag spark-integration-sample Search

Missing an organization? [Ensure Heroku Dashboard has organization access.](#)

ObjectIsAdvantag/spark-integration-sample Connect

Then click “Enable Automatic Deploys”.

You will end up with a Heroku deployment configuration similar to the one below:



App connected to GitHub

Code diffs, manual and auto deploys are available for this app.

Connected to ObjectIsAdvantag/spark-integration-sample

- ✓ Releases in the [activity feed](#) link to GitHub to view commit diffs
- ✓ Automatically deploys from master

Automatic deploys

Enables a chosen branch to be automatically deployed to this app.

Automatic deploys from master are enabled

Every push to master will deploy a new version of this app. Deploys happen automatically; be sure to always in a deployable state and any tests have passed before you push. [Learn more.](#)

☒ Wait for CI to pass before deploy

Only enable this option if you have a Continuous Integration service configured on your repo.

Disable Automatic Deploys

Now, click the black button “Deploy branch”, and watch Heroku build and deploy your code sample.

Manual deploy

Deploy the current state of a branch to this app.

Deploy a GitHub branch

This will deploy the current state of the branch you specify below. [Learn more](#).

master

Deploy Branch

Receive code from GitHub

Build master [Hide build log](#)

```
Procfile declares types --> (none)
Default types for buildpack --> web
-----> Compressing...
Done: 13.1M
-----> Launching...
Released v3
https://spark-integration-on-heroku.herokuapp.com/ deployed to Heroku
```

Build finished

Deploy to Heroku

Your app was successfully deployed.

[View](#)

When done, click View to open your OAuth Grant flow entry point. An HTML page such as the one below should show up.

Do not proceed and run the OAuth flow for now as we first need to register the integration to Cisco Spark.



Keep the Root URL of your integration in a safe place as we'll need it in the next step.

The Root URL in the example above is: <https://spark-integration-on-heroku.herokuapp.com/>


Step 3: Register your Cisco Spark Integration

The registration process is how your integration declares the set of authorizations required to execute its custom logic. It also specifies the redirect URL where our integration expects to receive the authorization code from Cisco Spark; from this authorization code, our integration will retrieve a time-limited access tokens, acting on behalf of the end-user.

Open the [“Create an Integration”](#) form on the "Spark for Developers" portal.

Fill in the form with a name, a description, an email and an icon for your integration. As the icon size must be 512x512 or larger and publicly accessible, you may interested in leveraging this [DevNet logo png](#).

In the redirect URI(s) field, paste the Heroku URL of your integration home page, and replace the index.html suffix by /oauth. In our example, the redirect URL is <https://spark-integration-on-heroku.herokuapp.com/oauth>.

Name	DevNet Learning Lab: "Register and deploy your own Spark Integration"
Description <small>What your integration will do in 256 characters or less</small>	This Spark integration fetches the end-user Cisco Spark account details and shows its Display Name. For that to happen, it initiates the OAuth Grant flow and asks for the "people:read" scope.
Support Email <small>Where users can get support for using your integration</small>	stsfartz@cisco.com
App Icon <small>A URL to an image that is 512x512 or larger is required</small>	https://cdn-images-1.medium.com/max/800/1*X00BUYYZLDk3f 
Redirect URI(s) <small>One or more URIs that a user will be redirected to when completing an OAuth</small>	https://spark-integration-on-heroku.herokuapp.com/oauth Add URI

The last field of the form concerns the OAuth scopes of your integration. These scopes correspond to the set of authorizations that the end-user will be asked to grant access for. For

the sake of this lab, we will specify that our integration can ask for all supported authorizations, and we'll refine these scopes [at runtime](#).

Check all scopes and click "Save changes".

Scopes

Scopes define the level of access that your integration requires. [Learn more](#)

- ☒ `spark:people_read` Read your users' company directory
- ☒ `spark:rooms_read` List the titles of rooms that your user's are in
- ☒ `spark:rooms_write` Manage rooms on your users' behalf
- ☒ `spark:memberships_read` List people in the rooms your user's are in
- ☒ `spark:memberships_write` Invite people to rooms on your users' behalf
- ☒ `spark:messages_read` Read the content of rooms that your user's are in
- ☒ `spark:messages_write` Post and delete messages on your users' behalf
- ☒ `spark:teams_read` List the teams your users are in
- ☒ `spark:teams_write` Create teams on your users' behalf
- ☒ `spark:team_memberships_read` List the people in the teams your user's belong to
- ☒ `spark:team_memberships_write` Add people to teams on your users' behalf

Cisco Spark automatically generates a Client ID and a Client Secret which we will use in the next step to configure our live integration running on Heroku. Keep this window opened or paste these values in a safe place.

Note that you won't need to copy the OAuth Authorization URL as the provided integration example [dynamically generates](#) this URL from the Client Id, Client Secret, Redirection URL, State and OAuth Scopes.

OAuth Settings

Learn more about authentication in the [Apps & OAuth Guide](#)

Client ID

C527f0bf8054082c353466b1216aab843d2170a86a6da9ed704bfaa2bf010924

Copy

Client Secret

02a63f1b56fae20d481579ff771950e9621b73a07dc1109edf04f562748c1

Copy

OAuth Authorization URL

You can use the URL below to initiate an OAuth permission request for this app. It is configured with your redirect URI and app scopes. Be sure to update the state parameter.

```
https://api.ciscospark.com/v1/authorize?client_id=C527f0bf8054082c353466b1216aab843d2170a86a6da9ed704bfaa2bf010924&response_type=code&redirect_uri=https%3A%2F%2Fspark-integration-on-heroku.herokuapp.com%2Foauth&scope=spark%3Amessages_write%20spark%3Arooms_read%20spark%3Ateams_read%20spark%3Amemberships_read%20spark%3Arooms_write%20spark%3Apeople_read%20spark%3Ams%20spark%3Amemberships_write%20spark%3Ateam_memberships_read%20spark%3Ateam_memberships_write%20spark%3Ateams_write&state=set_state_here
```

Step 4: Configure and test your Cisco Spark Integration

In this last step, we will configure our Cisco Spark integration running on Heroku.

Go to the settings tab of your Heroku project, and click “Reveal Env Variables”.

Add new variables for the CLIENT_ID, CLIENT_SECRET and REDIRECT_URI declared and generated in the previous step.

Note that you can also change the default State parameter of the OAuth Grant Flow by setting a STATE environment variable. Yet, in order to avoid HTTP replays, you will certainly want your code to [generate unique values for this State parameter](#). Moreover, you could associate this State with a back-end identifier in order to correlate the OAuth granted Access Token with a User Identity of your application.

Personal apps > spark-integration-on-heroku

GITHUB ObjectIsAdvantag/spark-integration-sample master

Overview Resources Deploy Metrics Activity Access Settings

Name spark-integration-on-heroku

Config Variables

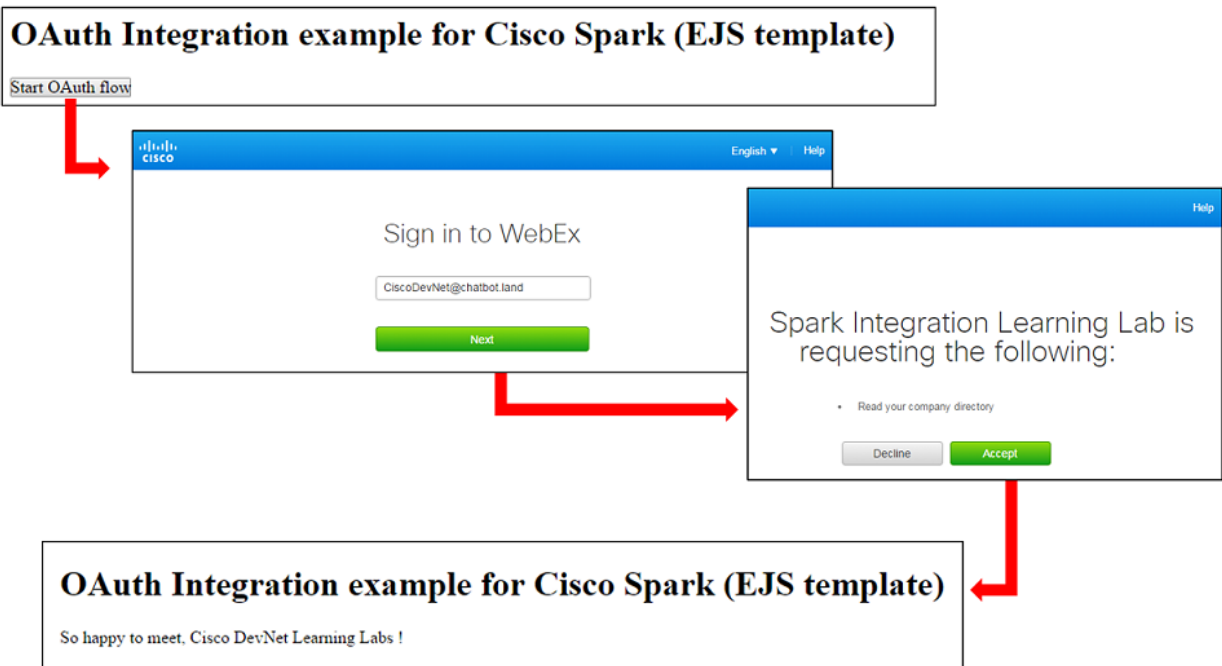
Config vars change the way your app behaves. In addition to creating your own, some add-ons come with their own.

KEY	VALUE
CLIENT_ID	C527f0bfb8054082c353466b1216aab843d2170a8
CLIENT_SECRET	02a63f1bfb6fae20d481579fff771950e9621b737
REDIRECT_URI	https://spark-integration-on-heroku.herokuapp.com

Open app

As you add or modify any "Config" variable, Heroku automatically restarts your integration so that you'll be happy to notice that your integration is now running live and ready to be invoked.

Click "Open app", and experiment your Cisco Spark Integration live.



To go further

What about making your integration now list the rooms of the Cisco Spark end-user that grants us permission to. Here are a few tips if you opt to implement this use case, edit the Github project you forked in step 1 and:

- change the OAuth permission with the ["spark:rooms_read" scope](#),
- switch the EJS template to the [rooms-list.ejs](#) template provided.
- And load the end-user's list of rooms by calling the [Spark API List Rooms](#) resource.