Cisco live!

June 25–29, 2017 · Las Vegas, NV

Extending the Collaboration Eco-System using Spark APIs for Non-Developers

Johannes Krohn, Technical Marketing Engineer

BRKCOL-2175

Cisco live!

Cisco Spark

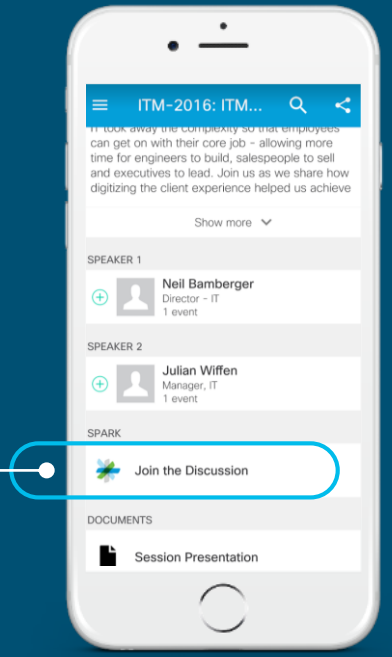
Questions?

Use Cisco Spark to communicate with the speaker after the session

How

1. Find this session in the Cisco Live Mobile App
2. Click “Join the Discussion”
3. Install Spark or go directly to the space
4. Enter messages/questions in the space

Cisco Spark spaces will be available until July 3, 2017.



cs.co/ciscolivebot#BRKCOL-2175

Abstract

Extensibility through open APIs is one of the key characteristics of Cisco Spark. This session will guide collaboration specialists without experience in SW development from zero to building the first simple integrations based on the Spark APIs.

The attendees will get an overview of the required concepts including REST, web services, APIs, bots, and integrations as well as an introduction to how to build web services using Python.

The session takes the attendees on the typical journey of a non-developer trying to make the Spark APIs productive while navigating through the cliffs of unknown concepts. This example helps the attendees to get over the 1st steep part of the learning curve more quickly.

Join the Cisco Spark Room for this session.

Agenda

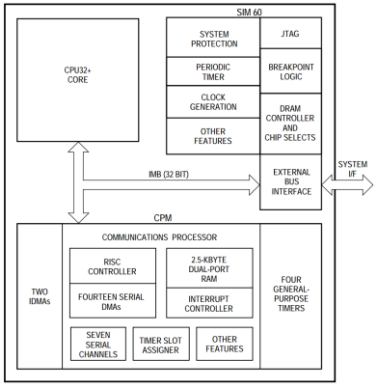
- Introduction
- Cisco Spark - APIs adding value
- The Concepts
- Cisco Spark APIs
- Tools
- Python
- Bots / Integrations
- Getting to Code
- Conclusion

About Me

Why is a collaboration TME talking about “coding”?

In and Out Of Coding

- Started coding in ‘82
  - Telefunken TR440 – Pascal, Fortran, ..
  - Apple II, C-64 – 65XX Assembler, Pascal, GALA (Game Language)
  - “Gepard” – 68k Assembler, Modula 2
- Studied Computer Science ’86 -
  - Pascal, 68k assembler, C/C++, Perl
  - The Web!
- ATM Networking SW developer ‘92-
  - 68k Assembler (MC68360), C/C++
- The “dark side” ‘98-
  - IT consulting
  - Cisco Sales
- Back to the Fun! 2007-
  - Working with APIs (AXL, etc.): Perl



- Playing with the “new” stuff 2014-
  - Learning Python
  - Spark APIs
  - Web Services

REST

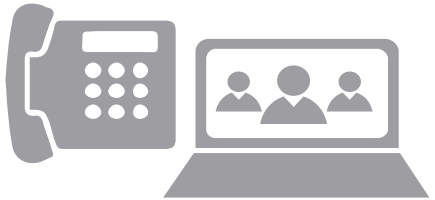
JSON

Cisco Spark – APIs Adding Value

Cisco Spark- Seamless Collaboration Experience

Meetings

Messaging



Calling[1]

[1] The Cisco Spark service doesn't include PSTN services. Customers need to purchase PSTN services from a 3rd party provider.
For the complete Cisco Spark service, Cisco preferred media provider ecosystem partners can provide PSTN local, long-distance, and direct-inward-dial services.
Existing Cisco UC customers will use Cisco Spark Hybrid Services to connect on-premises call capabilities to Cisco Spark capabilities in the cloud.

The Cisco Spark Platform

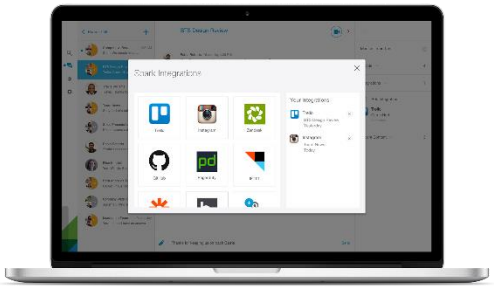
open and extensible

Make Spark the Place for All Your Work

Native Integrations

Easy to configure native integrations in the Cisco Spark app

pagerduty GitHub Instagram Trello zendesk



Integration Services

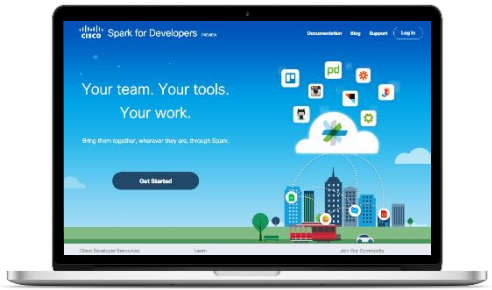
Integrate with other apps in seconds to automate tasks and make your life more efficient

Built.io Zapier IFTTT

Spark for Developers

Create custom integrations using the rich Cisco Spark APIs

developer.ciscospark.com



The Concepts

API, What, Where, and Why?

- Definition: .. is a set of **subroutine definitions**, **protocols**, and **tools** for building application software. In general terms, it is a set of **clearly defined methods of communication** between various software components. .. Documentation for the API is usually provided to facilitate usage."[1]
- APIs
  - Enabler for open systems integration
  - Universally available
  - Unleash developer innovation

[1]Wikipedia

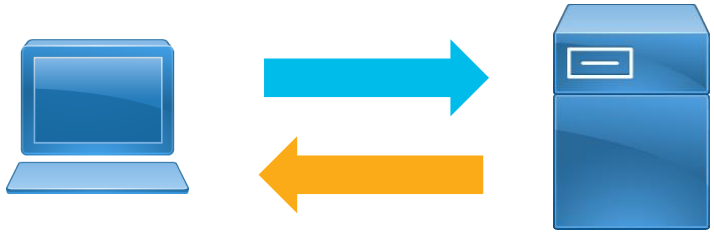
JSON – JavaScript Object Notation

- Open standard format
- Human-readable
- Language independent format (Python support through “json” module)
- Mime-type: application/json
- Object: unordered set of name/value pairs
- Data types:
  - Number
  - String
  - Boolean
  - Array
  - Object


```
{
   "created": "2012-06-15T20:51:06.512Z",
   "displayName": "Johannes Krohn",
   "orgId": "Y2lzY29…C1hZDcyY2FlMGUxMGY",
   "firstName": "Johannes",
   "type": "person",
   "lastName": "Krohn",
   "id": "Y2l…MTMyOTk",
   "emails": [
      "jkrohn@cisco.com"
   ],
   "status": "active",
   "nickName": "Johannes"
}
```

REST – Representational State Transfer

- Not really a standard – more an architecture
- Uses existing standards: for example HTTP(S) for transport
- All about client-server
- Conceptually similar to web browser accessing web server
- Resources
  - Every resource can be addressed by a URI
  - Methods: GET, PUT, POST, DELETE, HEAD
  - Uniform representation: typically JSON
- Protocol
  - Stateless
  - Client-server



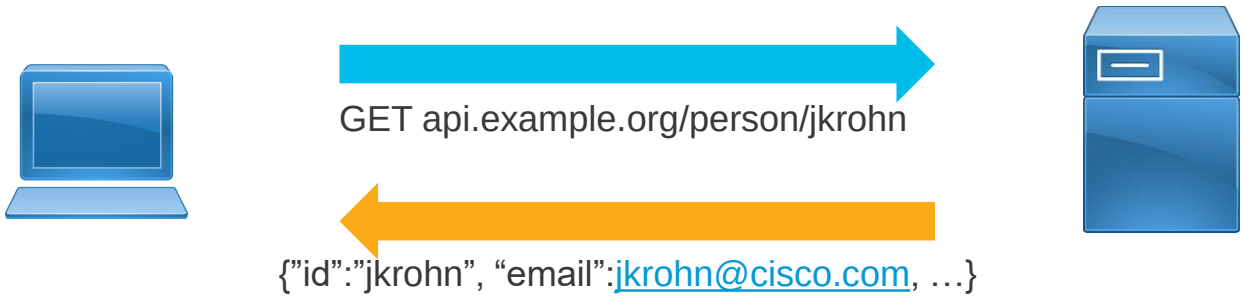
REST Methods

- CRUD: four basic operations of persistent storage
  - Create, Read, Update, Delete
- Mapped to HTTP methods in REST

Operation	HTTP Method in REST
Create	POST
Read	GET
Update	PUT / POST
Delete	DELETE

Example: Read Information

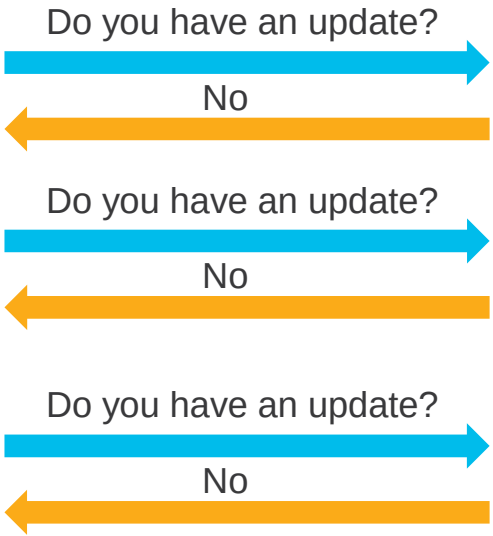
- GET to read information via REST API
- .. equivalent to a web browser retrieving information from a web server
- Information returned as JSON data



Webhooks – Problem Statement

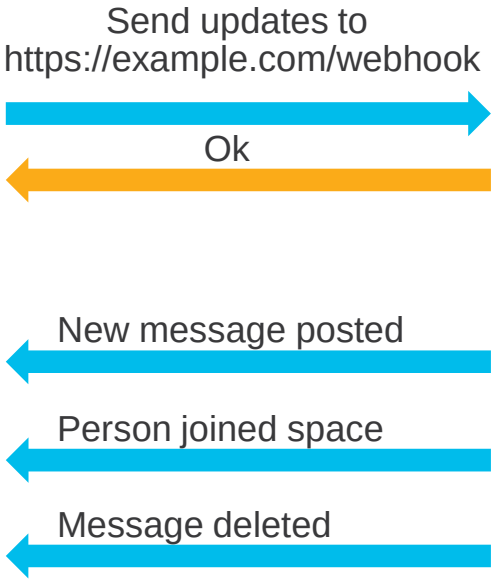
- Polling for events is inefficient and does not scale
- Too many instances polling
- Too many event types to poll for

  → not really an option



Webhooks - Concept

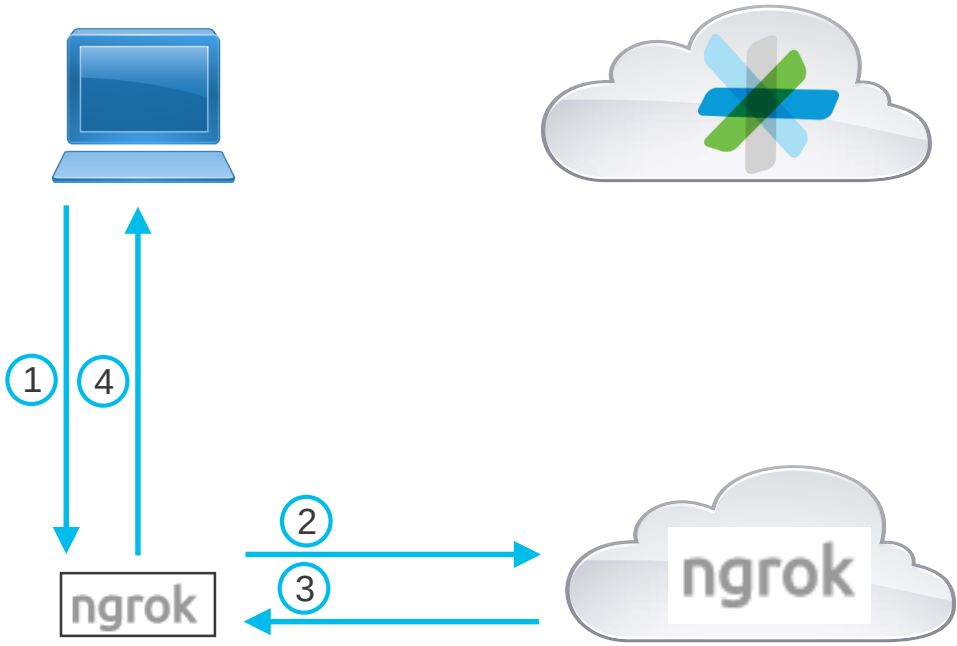
- Ask for notifications
- Register Webhook
  - HTTP callback
- Web service “calls” Webhook
  - POST to registered URL
- Publish/Subscribe instead of Polling
- Requires public URL for callbacks



Webhooks w/o Public URI

- What if code runs on host not publicly reachable?
  - Inside firewall
  - No public hostname
- Ngrok: cloud service to tunnel public URL to host
- Ngrok client on host creates persistent connection
- Ngrok client on host relays requests received from the cloud to localhost

1. Start ngrok client
2. Create persistent connection
3. Obtain public URL
4. Report public URL

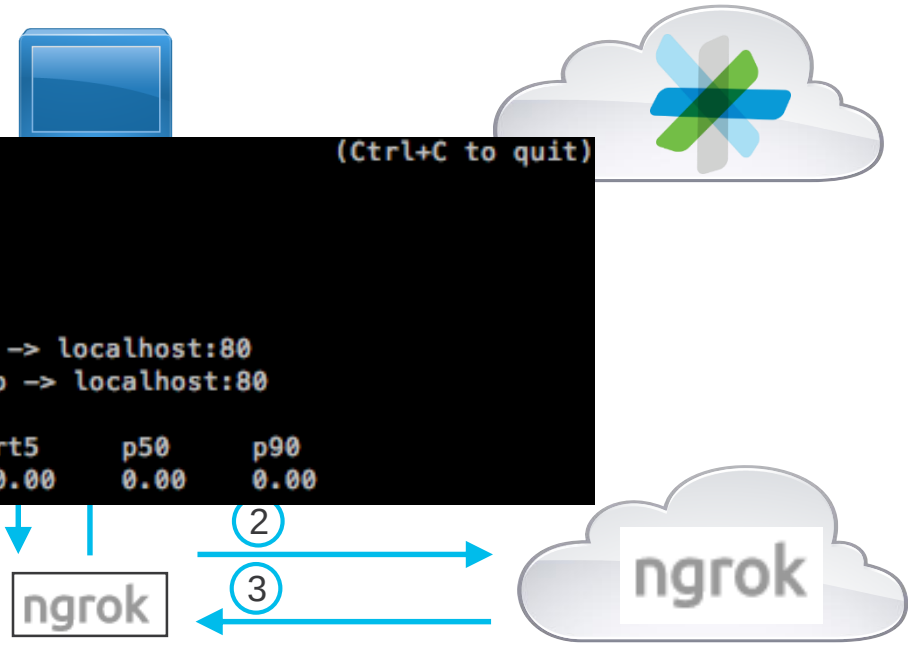


Webhooks w/o Public URI

- What if code runs on host not publicly reachable?
  - Inside firewall
  - No public hostname

```
ngrok by @inconshreveable                                       (Ctrl+C to quit)

Session Status                online
Version                       2.2.4
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://da96faa6.ngrok.io -> localhost:80
Forwarding                    https://da96faa6.ngrok.io -> localhost:80

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

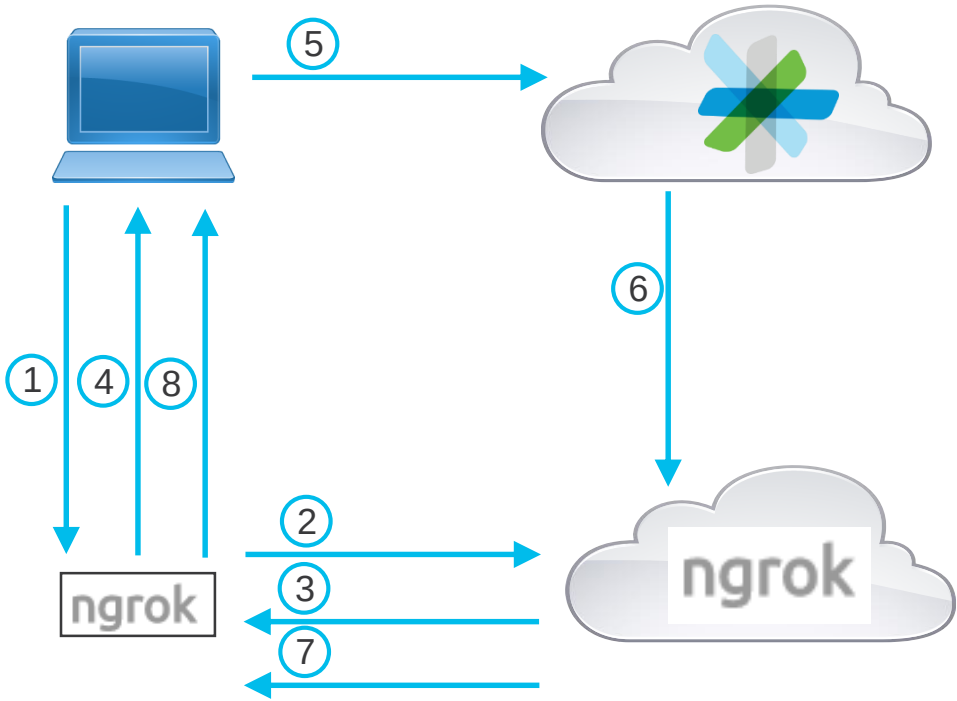
2. Create persistent connection
3. Obtain public URL
4. Report public URL



Webhooks w/o Public URI

- What if code runs on host not publicly reachable?
  - Inside firewall
  - No public hostname
- Ngrok: cloud service to tunnel public URL to host
- Ngrok client on host creates persistent connection
- Ngrok client on host relays requests received from the cloud to localhost

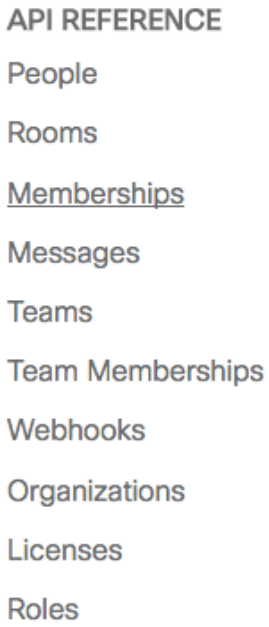
1. Start ngrok client
2. Create persistent connection
3. Obtain public URL
4. Report public URL
5. Create webhook w/ public URL
6. POST to public URL
7. Relay via persistent connection
8. POST to localhost



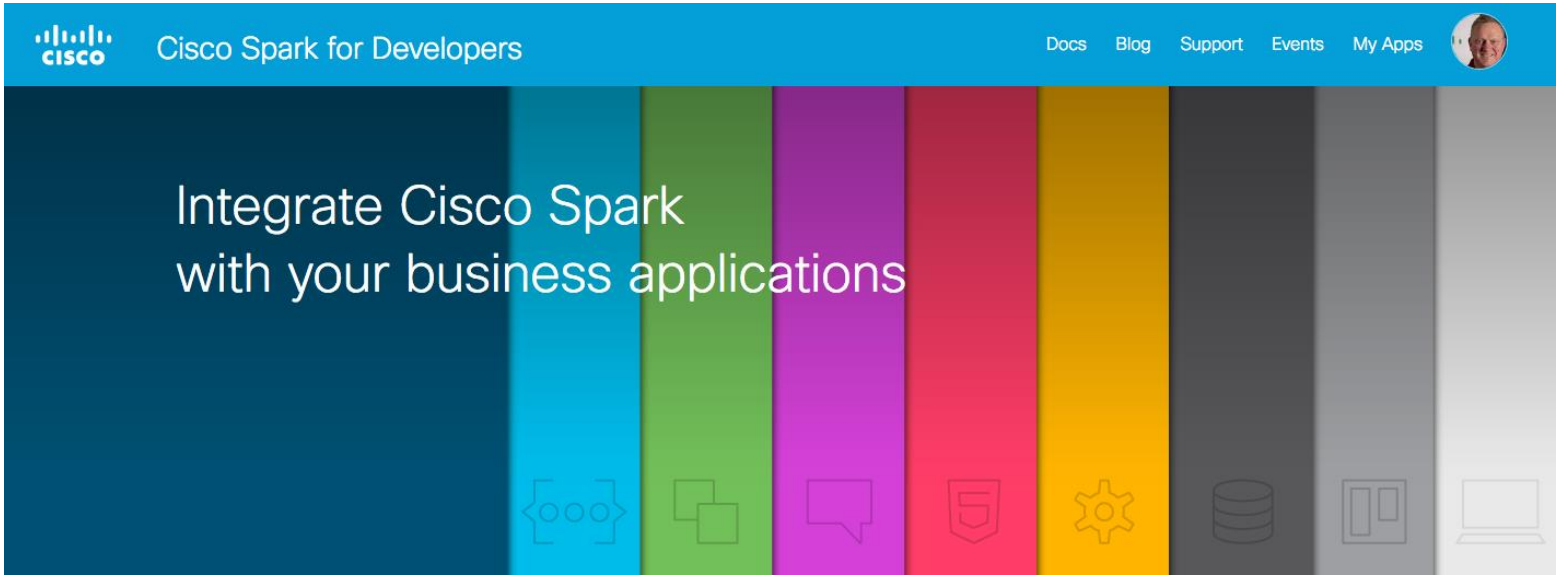
Cisco Spark APIs

Cisco Spark APIs

- Documentation: http://developer.ciscospark.com
- Objects
  - People
  - Rooms/Spaces
  - Memberships
  - Messages
  - Teams
  - Team Memberships
  - Webhooks
  - Organizations
  - Licenses
  - Roles
- OAuth access token used for authorization

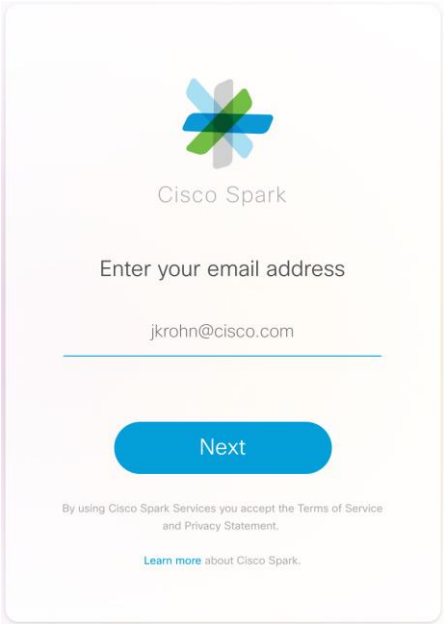


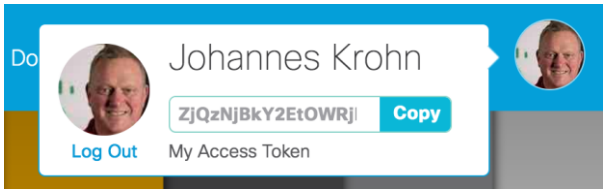
Demo: Cisco Spark API Documentation



Developer.cisco.com – Login, OAuth token

- Login using your Cisco Spark account
- OAuth access token is generated

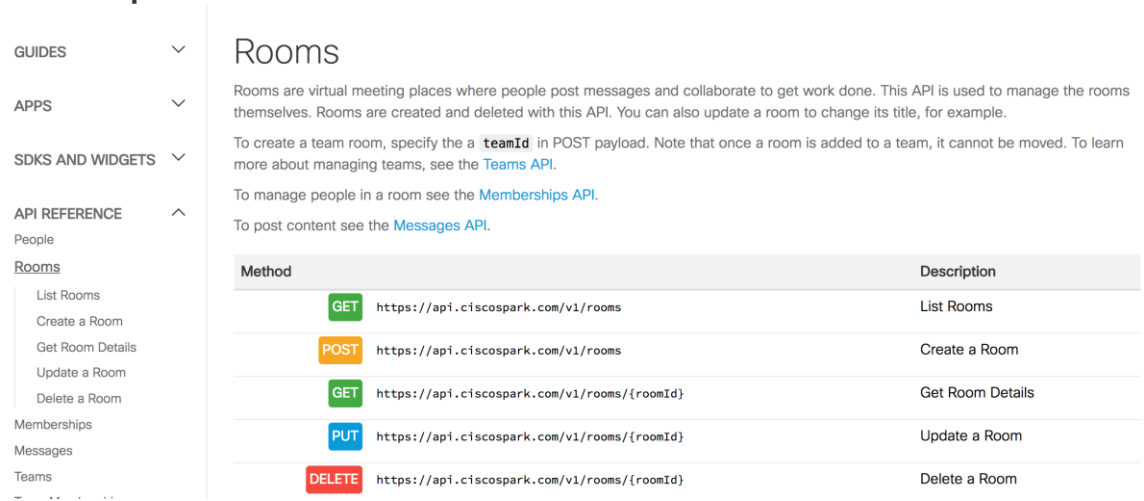


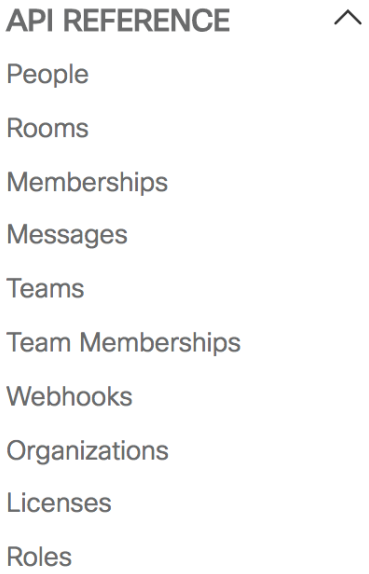


Developer.cisco.com – API Doc

- API Reference lists all available APIs
- For example ”Rooms” APIs for operations on spaces

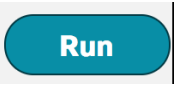
Docs

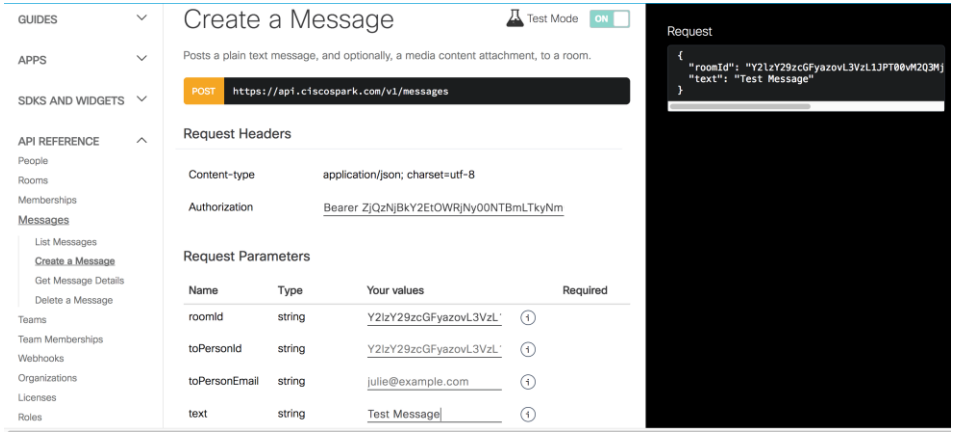


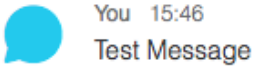


Developer.cisco.com – Live Method Execution

- With “Test Mode” activated API calls can be executed live
- List Spaces to get space ID
- Create Message in space using that space ID
- Direct Testing of APIs



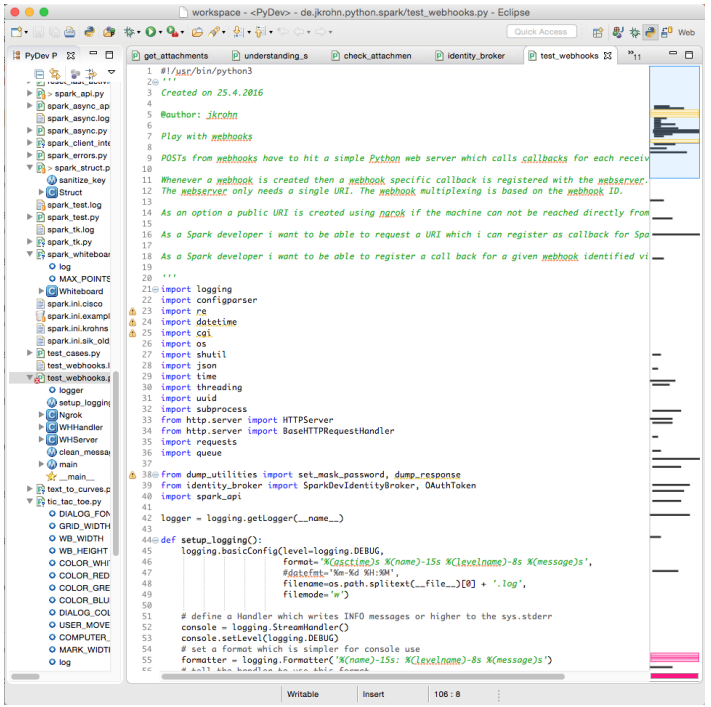




Tools

IDE – Integrated Development Environment

- Helps to develop and test your application
- Features
  - GUI
  - Editor
  - Build automation
  - Syntax highlighting
  - Debugger
  - Integration w/ revision control system (e.g. Git)



Syntax Highlighting

- What Do you prefer?
- This?

```
def get_attachments():

    def assert_folder(p_state, base_path, room_id, room_folder):
        ''' make sure that the folder is created for the room
        '''
        if not os.path.lexists(base_path):
            # base directory needs to be created
            logging.debug('Base directory %s does not exist' % base_path)
            os.mkdir(base_path)

        full_path = os.path.join(base_path, room_folder)

        if room_id not in p_state:
            p_state[room_id] = {}
        room_state = p_state[room_id]

        if 'folder' not in room_state:
            logging.debug('No previous folder for room %s' % room_folder)
            # the folder for this room hasn't been created before
            i = 0
            base_folder = room_folder
            while True:
                full_path = os.path.join(base_path, room_folder)
                try:
                    os.mkdir(full_path)
                    logging.debug('Created folder %s' % full_path)
                except FileExistsError:
```

Syntax Highlighting

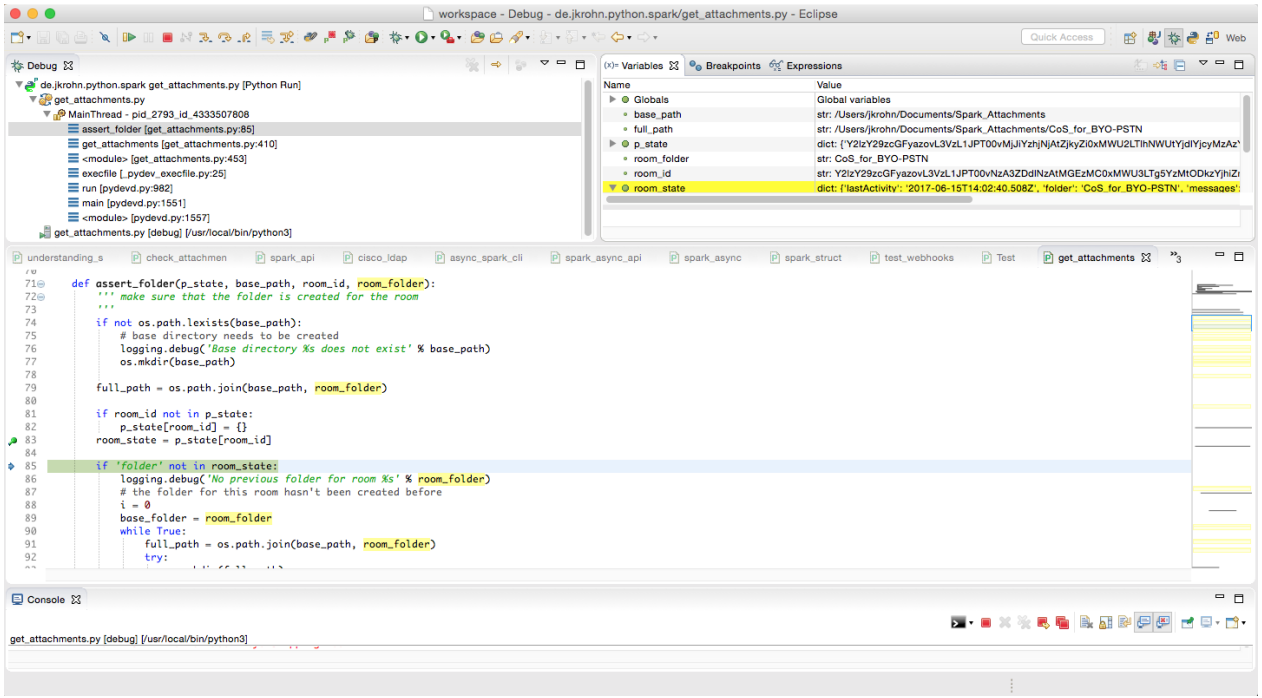
- What Do you prefer?
- Or this?

```
def get_attachments():

    def assert_folder(p_state, base_path, room_id, room_folder):
        ''' make sure that the folder is created for the room
        '''
        if not os.path.lexists(base_path):
            # base directory needs to be created
            logging.debug('Base directory %s does not exist' % base_path)
            os.mkdir(base_path)

        full_path = os.path.join(base_path, room_folder)

        if room_id not in p_state:
            p_state[room_id] = {}
        room_state = p_state[room_id]

        if 'folder' not in room_state:
            logging.debug('No previous folder for room %s' % room_folder)
            # the folder for this room hasn't been created before
            i = 0
            base_folder = room_folder
            while True:
                full_path = os.path.join(base_path, room_folder)
                try:
                    os.mkdir(full_path)
                    logging.debug('Created folder %s' % full_path)
                except FileExistsError:
```

Live Debugger

- Live Debugger allows to
  - Set breakpoints
  - Check variables
  - Evaluate expressions

→ Essential for effective SW development



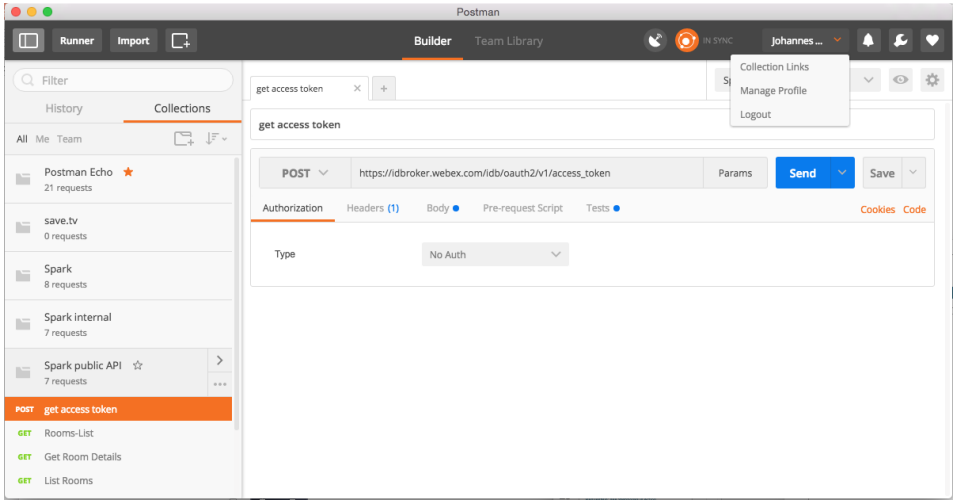
IDEs for Python

- IDLE (Standard IDE)
- PyCharm
- PyDev in Eclipse
- PythonAnywhere
- Cloud9

PyCharm

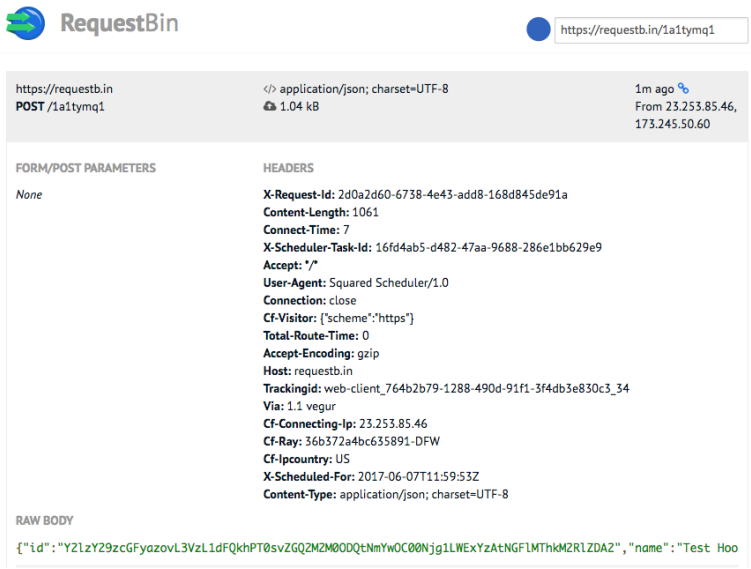
Postman: Test APIs

- Share, test, document & monitor APIs
- Easily test API calls
- Generate code (Python, curl, ..)
- Create collections
- Available for Mac, Windows, Linux, and Chrome apps
- https://www.getpostman.com/



RequestBin: See Webhooks in Action

- Free service: http://requestb.in
- Creates unique URL
- Use case: Cisco Spark webhook pointing to Requestb.in to test webhook operation
- Provides real-time view on requests hitting the URL



Github

- Git repository hosting service
- Offers
  - Revision control
  - Source code management
- THE place to share your code

Python

Python – The Language

- Friendly and easy to learn, pleasant
- Readable
- Open
- Free
- Runs everywhere
- Flexible
- Fast
- Powerful; Modules for everything! (https://pypi.python.org/pypi)

Python Characteristics

- Multi-purpose; not only “scripting”
  - GUI
  - Web development
  - Apps
  - ..

- Interpreted language .. actually compiled byte-code is executed
- Object Oriented
- Strongly typed
- Widely used: https://www.python.org/about/success/

Python Releases

- History
  - 1989: created by Guido Van Rossum
  - 1994: Python 1.0 released

    ..
  - 2000: Python 2.0 released – latest 2.x release is 2.7; released 2010

    ..
  - 2008: Python 3.0 released – broke backward compatibility

    ..
  - 2015: Python 3.5 released
  - 2016: Python 3.5 released
- New feature development only in 3.x
- Recommendation: start with latest Python release (3.6)

Learning Python

- Beginner's Guide: https://wiki.python.org/moin/BeginnersGuide
- The Hitchhiker's Guide to Python: http://python-guide-pt-br.readthedocs.io/en/latest/intro/learning/
- Online Courses
  - https://www.edx.org/course/subject/computer-science/python
  - https://www.coursera.org/specializations/python
  - https://www.codecademy.com/learn/python
- Great Book: "Learning Python, 5th Edition", Mark Lutz; PDF available online
- Start with fun stuff (Sudoku?, ..)
- Code, Play, have fun!

Executing Python Code

- Interactive: simply start python from the CLI

```
~ jkrohn$ python3
Python 3.5.0 (v3.5.0:374f501f4567, Sep 12 2015, 11:00:19)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello world!')
Hello world!
```

- Run python file from the CLI

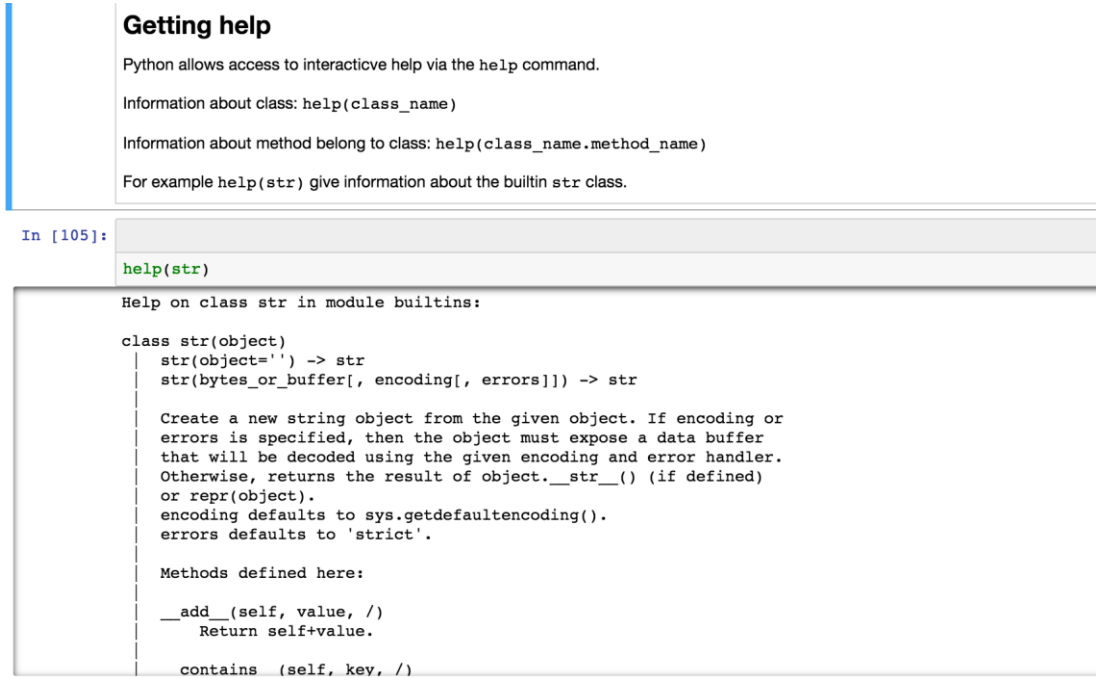
```
~ jkrohn$ python hello_world.py
Hello World!
```

- Demos: Interactive Jupyter Notebooks
  - “The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and explanatory text”

Demo – Python Basics

```
>>>
>>> s = "hello"
>>> s
'hello'
>>> s = 1
>>> s
1
>>> dict = {'name':'Bob', 'age':35, 'sex':'male'}
>>> dict
{'age': 35, 'sex': 'male', 'name': 'Bob'}
>>> dict['age']
35
>>> import json
>>> print(json.dumps(dict, indent=4))
{
    "age": 35,
    "sex": "male",
    "name": "Bob"
}
>>>
```

Getting Help



Getting help

Python allows access to interacticve help via the `help` command.

Information about class: `help(class_name)`

Information about method belong to class: `help(class_name.method_name)`

For example `help(str)` give information about the builtin `str` class.

In [105]:

```
help(str)
```

```
Help on class str in module builtins:

class str(object)
 |  str(object='') -> str
 |  str(bytes_or_buffer[, encoding[, errors]]) -> str
 |
 |  Create a new string object from the given object. If encoding or
 |  errors is specified, then the object must expose a data buffer
 |  that will be decoded using the given encoding and error handler.
 |  Otherwise, returns the result of object.__str__() (if defined)
 |  or repr(object).
 |  encoding defaults to sys.getdefaultencoding().
 |  errors defaults to 'strict'.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
```

Getting Help

If you want detailed information about one specific method then you could use: `help(str.startswith)`

In [106]:
```
help(str.startswith)
```

```
Help on method_descriptor:

startswith(...)
    S.startswith(prefix[, start[, end]]) -> bool

    Return True if S starts with the specified prefix, False otherwise.
    With optional start, test S beginning at that position.
    With optional end, stop comparing S at that position.
    prefix can also be a tuple of strings to try.
```

In [107]:
```
'my string'.startswith('my')
```

Out[107]: True

In [108]:
```
'my string'.startswith('xyz')
```

Out[108]: False

Comments

Comments

Any line starting with # is considered a comment and the text following the # is ignored during code execution

Multiline comments start and end with: '''

In [109]:

```
# this is a comment

''' this is a comment
    spanning multiple lines
'''

# finally some code
print('Hello world!')
```

```
Hello world!
```

Variables

Variables

Variables in Python are named locations which store references to objects stored in memory.

You can assign any value to any variable. Types don't need to be declared. The type of a variable is automatically detected at runtime based on the type of the value you assign.

In [110]:

Edit Attachments

```
v = 'hello'
type(v)
```

Out[110]: str

In [111]:

Edit Attachments

```
v = 10
type(v)
```

Out[111]: int

In [112]:

Edit Attachments

```
v = 10.0
type(v)
```

Out[112]: float

Simultaneous Assignments

Simultaneous Assignments

Python does not only have simple assignments (see above), but also allows to assign multiple values to multiple varibles at the same time

In [113]:

```
a, b, c = 1, 2, 3
print('a = {}, b = {}, c = {}'.format(a, b, c))
```

```
a = 1, b = 2, c = 3
```

This can be used to simply swap values between two variables w/o using an intermediate variable. Instead of

```
a = 1
b = 2

# now swap
t = a
a = b
b = t
```

we can simply:

In [114]:

```
a = 1
b = 2
# now swap
a, b = b, a
print('a = {}, b = {}'.format(a, b))
```

```
a = 2, b = 1
```

Data Types

Data Types

Python has five standard types:

1. Numbers (int, float, ..)
2. String
3. List
4. Tuple
5. Dictionary
6. Boolean - values are **`True`** and **`False`**, but also other values are considered **`False`**:

    A. 0, 0.0
    B. empty List - []
    C. empty Tuple - ()
    D. empty Dictionary - {}
    E. **None**

Strings

Strings

Strings can be indexed and sliced

In [115]:

```
a = 'abcdefghijkl'
print(a[0])
print(a[2])
```

```
a
c
```

Negative indexes start at the end

In [116]:

```
print(a[-1])
print(a[-4])
```

```
l
i
```

Slicing provides access to parts of a string

In [117]:

```
print(a[1:])   # everything starting at the second character
print(a[:3])   # 1st three characters
print(a[2:5])  # starting at the 3rd character until (including) the 5th
print(a[2:-4]) # starting at the 3rd character until (not including) the 4th from the end
```

```
bcdefghijkl
abc
cde
cdefgh
```

List and Tuples

Lists and tuples

Lists and tuples can be indexed as well.

In [118]:

```
some_list = [1, 2, 3, 'a', 4]
some_tuple = (1, 2, 3, 'a', 4)


print(some_list[2])
print(some_tuple[2])
```

```
3
3
```

Slicing allows to access parts of lists and tuples.

In [119]:

```
print(some_list[1:])
print(some_tuple[1:])
```

```
[2, 3, 'a', 4]
(2, 3, 'a', 4)
```

Mutable vs. Immutable

But only list elements can be updated. List are **mutable** while tuples are **immutable**.

In [120]:
```
some_list[2] = 'new value'
print(some_list)
```

```
[1, 2, 'new value', 'a', 4]
```

In [121]:
```
some_tuple[2] = 'new value'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-121-2f3d3ad242b9> in <module>()
----> 1 some_tuple[2] = 'new value'

TypeError: 'tuple' object does not support item assignment
```

Dictionaries

Dictionaries

A dictionary allows to store multiple values each under a unique key. The key can be of any **immutable** type.

In [122]:

```
bob = {'name' : 'Bob', 'email':'bob@example.com'}
alice = {'name' : 'Alice', 'email':'alice@example.com'}

print(bob['name'])

# dictionary holding both entries indexed by email
persons = {bob['email'] : bob, alice['email'] : alice }

print(persons['bob@example.com'])
```

```
Bob
{'name': 'Bob', 'email': 'bob@example.com'}
```

Modules

Modules

Modules can be imported using:

```
import some_module_to_import
```

You can import multiple modules in one statement:

```
import some_module_to_import, another_module_to_import
```

In [123]:

```
import math
```

In [124]:

```
print(math.pi)
```

```
3.141592653589793
```

Handling JSON

Handling JSON

The `json` module has all the tools to handle JSON.

Create JSON string from Python variable

`json.dumps()` allows to dump a Python variable (typically a dict or a list) to a string.

In [125]:

Edit Attachments

```
import json

bob = {'name' : 'Bob', 'email':'bob@example.com'}
alice = {'name' : 'Alice', 'email':'alice@example.com'}

persons = {bob['email'] : bob, alice['email'] : alice }
print(json.dumps(persons))
```

```
{"bob@example.com": {"name": "Bob", "email": "bob@example.com"}, "alice@example.com": {"name": "Alice", "email": "ali
ce@example.com"}}
```

Handling JSON

Note: keys and strings in JSON use " as quotes

`json.dumps()` also provides an easy way to "pretty print" json data.

Hint: used heavily when working with JSON data

In [126]:

```
print(json.dumps(persons, indent=4))
```

```
{
    "bob@example.com": {
        "name": "Bob",
        "email": "bob@example.com"
    },
    "alice@example.com": {
        "name": "Alice",
        "email": "alice@example.com"
    }
}
```

Handling JSON

Evaluate JSON string and store the value into Python variable

`json.loads` allows to interpret the contents of a string as JSON and put the resulting object into a Pyhton variable

In [127]:

Edit Attachments

```
json_string = '{"bob@example.com": {"name": "Bob", "email": "bob@example.com"}, "alice@example.com": {"name": "Alice",

data = json.loads(json_string)
data
```

Out[127]:

```
{'alice@example.com': {'email': 'alice@example.com', 'name': 'Alice'},
 'bob@example.com': {'email': 'bob@example.com', 'name': 'Bob'}}
```

Handling JSON

Invalid JSON data will raise a `json.JSONDecodeError` exception.

In [128]:

Edit Attachments

```
json_string = "{'bob@example.com': {'name': 'Bob', 'email': 'bob@example.com'}, 'alice@example.com': {'name': 'Alice',
data = json.loads(json_string)
```

```
---------------------------------------------------------------------------
JSONDecodeError                           Traceback (most recent call last)
<ipython-input-128-5584468584ba> in <module>()
      1 json_string = "{'bob@example.com': {'name': 'Bob', 'email': 'bob@example.com'}, 'alice@example.com': {'name':
 'Alice', 'email': 'alice@example.com'}}"
----> 2 data = json.loads(json_string)

/Library/Frameworks/Python.framework/Versions/3.5/lib/python3.5/json/__init__.py in loads(s, encoding, cls, object_ho
ok, parse_float, parse_int, parse_constant, object_pairs_hook, **kw)
    317             parse_int is None and parse_float is None and
    318             parse_constant is None and object_pairs_hook is None and not kw):
--> 319         return _default_decoder.decode(s)
    320     if cls is None:
    321         cls = JSONDecoder
```

Handling JSON

What went wrong here?

A typical pattern when reading JSON data is to intercept the `json.JSONDecodeError` exception.

In [129]:

```
try:
    data = json.loads(json_string)
except json.JSONDecodeError:
    print('JSON string failed to parse')
    data = None

print(data)
```

```
JSON string failed to parse
None
```

Bots / Integrations

BOT

- Intelligent software agent
- Acting as ”individual”; act on their own behalf
- Machine accounts to
  - Automate routine tasks
  - Participate in Spark conversations
- Typical types of bots:
  - Notifier: post notifications to Spark spaces
  - Controller: text based remote control (“find info”)
  - Assistant: natural language processing, answer questions etc.
- Bots only have access to Cisco Spark messages they are “@” mentioned in

Integration

- Act on behalf of a Cisco Spark user
  - Access equivalent to a real spark User (limited by authorized scopes)
- Invoke Cisco Spark APIs on behalf of user
- Requires authorization of integration by user
  - OAuth Grant Flow to authenticate user and ask for authorization
  - User approves authorization levels (scopes) requested by the integration
- Each Integration has a client ID, client secret and redirect URI
- Documentation: https://developer.ciscospark.com/authentication.html

oAuth Flow Summary

1. Application Requests *auth code*
Browser redirect to Spark Authentication

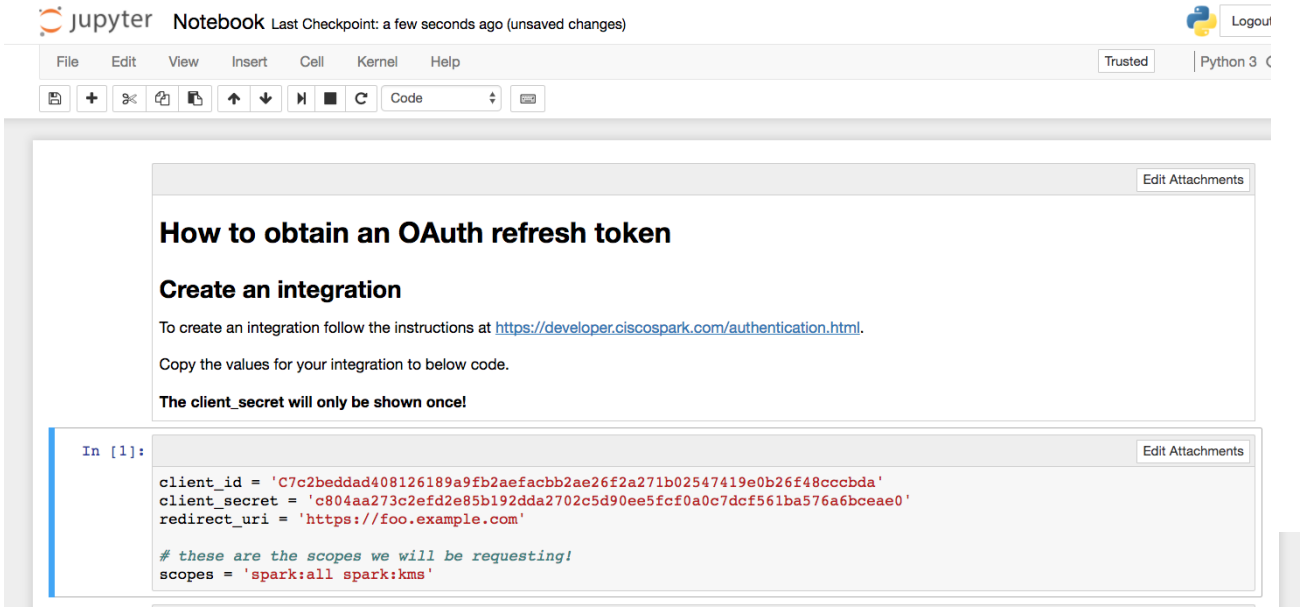
2. Spark returns the *auth code* to application
Browser redirect to Application

3. Request an *access token*
HTTP GET request to Spark API

4. Application gets *access token* and *refresh token*
HTTP GET response from Spark API

Demo: Integration Authorization

- View the static notebooks at https://github.com/jeokrohn/brkcol2175clus2017/tree/master/Notebook



https://goo.gl/MbFC9k

Demo: Integration Authorization

How to obtain an OAuth refresh token

Create an integration ¶

To create an integration follow the instructions at https://developer.ciscospark.com/authentication.html.

Every integration needs a public URL defining the icon of the integration.

Example: https://cdn2.iconfinder.com/data/icons/business-seo-vol-5/100/1-07-512.png

Copy the values for your integration to below code.

The client_secret will only be shown once!

In this demo we are using an non-existing redirect URI so that the OAuth flow will stall at the point where the browser finally tries to access the redirect URI.

The example code tries to read the client secret from an environment variable `SPARK_CLIENT_SECRET` first, so that i don't have to share my client secret publicly :-)

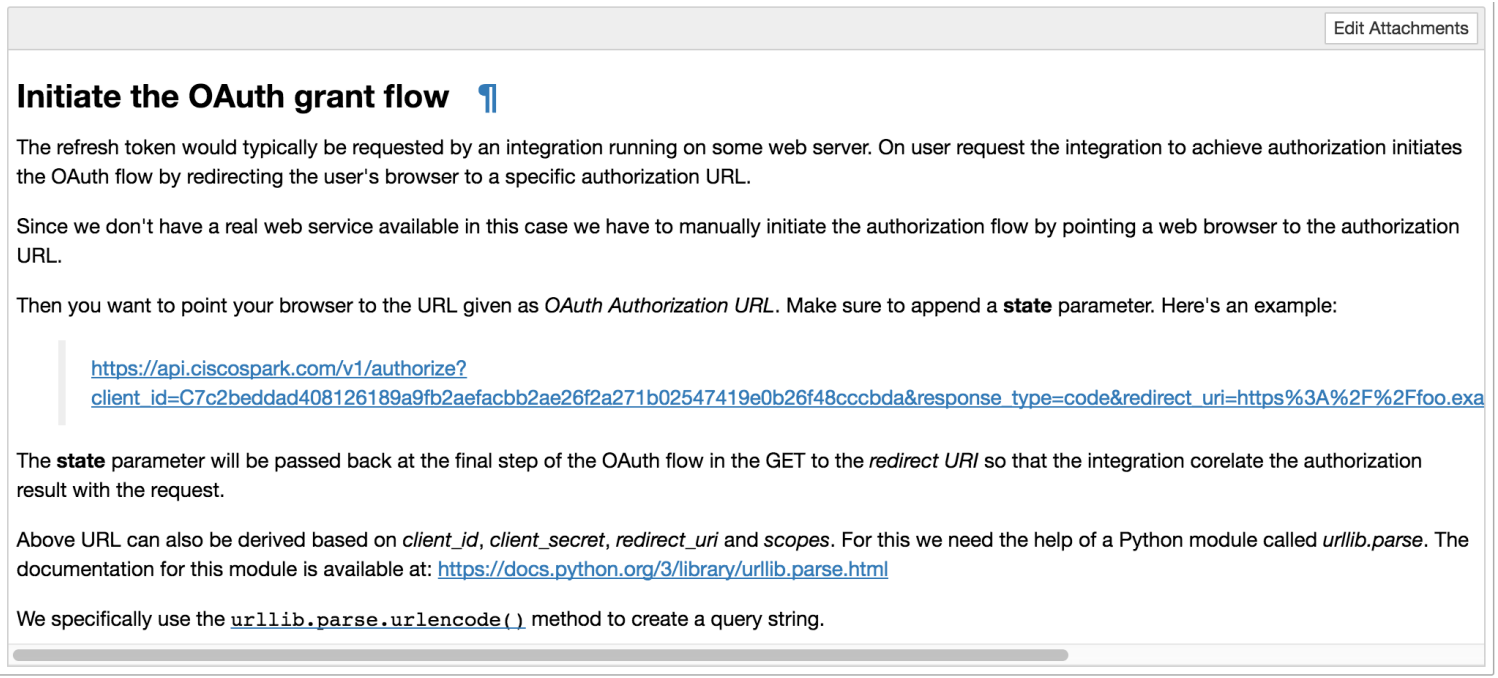
Demo: Integration Authorization

In [1]:

Edit Attachments

```
import os

client_id = 'C7c2beddad408126189a9fb2aefacbb2ae26f2a271b02547419e0b26f48cccbda'
redirect_uri = 'https://foo.example.com'
client_secret = os.environ.get('SPARK_CLIENT_SECRET')

if client_secret is None:
    client_secret = '<insert your client secret here>'


# these are the scopes we will be requesting!
scopes = 'spark:all spark:kms'
```

Demo: Integration Authorization



Demo: Integration Authorization

In [2]:

Edit Attachments

```
# we need the help from an external Python libraries
import urllib.parse
import uuid

base_url = 'https://api.ciscospark.com/v1/authorize'

state = str(uuid.uuid4())   # some random UUID
print('State: {}'.format(state))

# prepare a dictionary with all parameters we need to encode in the URL
data = {
    'client_id' : client_id,
    'response_type' : 'code',
    'redirect_uri' : redirect_uri,
    'scope' : scopes,
    'state' : state
}

# this gets us a url encoded query string
query = urllib.parse.urlencode(data, quote_via=urllib.parse.quote)

# .. which we then finally combine with the base url
url = '{}?{}'.format(base_url, query)
print(url)
```

```
State: e6b80cca-712a-49cc-afde-ef77fa9370a8
https://api.ciscospark.com/v1/authorize?scope=spark%3Aall%20spark%3Akms&response_type=code&state=e6b80cca-712a-49cc-afde-ef77fa9370a8&client_id=C7c2beddad408126189a9fb2aefacbb2ae26f2a271b02547419e0b26f48cccbda&redirect_uri=https%3A%2F%2Ffoo.example.com
```

Demo: Integration Authorization

Edit Attachments

Exchange *code* for a refresh token

Since we defined an non-existant redirect URL the browser fails to follow the ultimate 302 redirect in the authorization flow. In reality the redirect URL would point to the webservice which initiated the authorization flow. Encoded in the URL is the *code* parameter which the web service would exchange for an OAuth refresh token.

Paste the address from your webbrowser into the last_redirect variable in below code. The code then extracts the *code* from the URL.

In [4]:

Edit Attachments

```
last_redirect = 'https://foo.example.com/?code=YmFhYzVkZmEtMTJmZC00YjIzLTg4NTQtNDliZWVkYjYwZTEzMTY0MWE2MTktZTUy&state=e(

# 1st we parse the URL. We are only interested in the query string
query = urllib.parse.urlparse(last_redirect).query

# then we parse the query string and get a dictionary with key/value pairs
query = urllib.parse.parse_qs(query)

# from that dictionary we finally extract the code
code = query['code'][0]
state = query['state'][0]

print('Code: {}'.format(code))
print('State (same as in the request above!): {}'.format(state))
```

```
Code: YmFhYzVkZmEtMTJmZC00YjIzLTg4NTQtNDliZWVkYjYwZTEzMTY0MWE2MTktZTUy
State (same as in the request above!): e6b80cca-712a-49cc-afde-ef77fa9370a8
```

Demo: Integration Authorization

Edit Attachments

With that **`code`** the web service (in our case that again is a manual process) can now get a refresh token by exchanging the **`code`** for a token. The **`code`** can only be used **once**.

To authorize the request to exchange the **`code`** for a refresh token the integration has to include the **`client_secret`** in the request. This is the only time the **`client_secret`** is used.

This is done by executing a POST agains a well-known web service as documented here: https://developer.ciscospark.com/authentication.html

Demo: Integration Authorization

In [ ]:

Edit Attachments

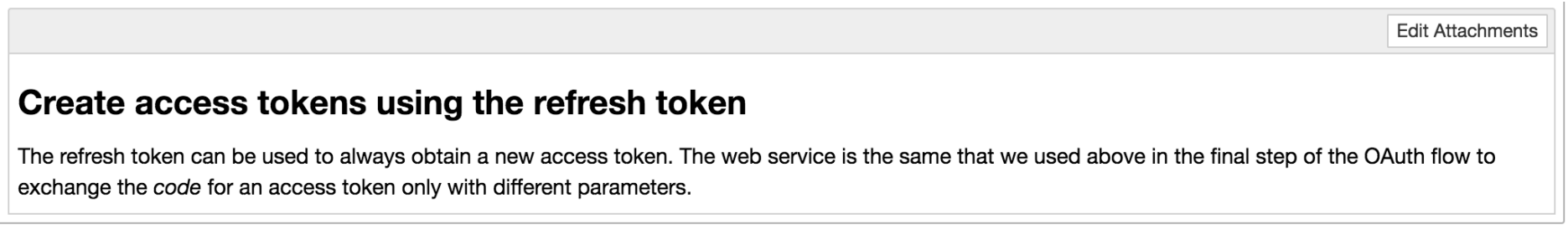
```
import requests
import json

access_token_url = 'https://api.ciscospark.com/v1/access_token'

params = {
    'grant_type' : 'authorization_code',
    'client_id' : client_id,
    'client_secret' : client_secret,
    'code' : code,
    'redirect_uri' : redirect_uri
}

r = requests.post(access_token_url, json = params).json()

print('JSON response:')
print(json.dumps(r, indent=4))

if r.get('errors'):
    error = r['errors'][0]
    print('Failed to get access token: {}'.format(error['description']))
else:
    access_token = r['access_token']
    refresh_token = r['refresh_token']

    print('\nAccess token: {}'.format(access_token))
    print('Valid for {} days'.format(round(r['expires_in'] / 60 / 60 / 24)))
    print('Refresh token: {}'.format(refresh_token))
    print('Valid for {} days'.format(round(r['refresh_token_expires_in'] / 60 / 60 / 24)))
```

Demo: Integration Authorization

```
if r.get('errors'):
    error = r['errors'][0]
    print('Failed to get access token: {}'.format(error['description']))
else:
    access_token = r['access_token']
    refresh_token = r['refresh_token']

    print('\nAccess token: {}'.format(access_token))
    print('Valid for {} days'.format(round(r['expires_in'] / 60 / 60 / 24)))
    print('Refresh token: {}'.format(refresh_token))
    print('Valid for {} days'.format(round(r['refresh_token_expires_in'] / 60 / 60 / 24)))
```

```
JSON response:
{
    "refresh_token": "OG                                    tMTBhOTcwM2QxN2M2YWIyZGNiMmUtNmI5",
    "expires_in": 1209599,
    "refresh_token_expires_in": 7702688,
    "access_token": "ZjIyODI                                        NlNjAxYmI3MjAtNzA3"
}

Access token: ZjIyOD                                           xYmI3MjAtNzA3
Valid for 14 days
Refresh token: OGY2MjN                                          GNiMmUtNmI5
Valid for 89 days
```

Demo: Integration Authorization



Demo: Integration Authorization

In [ ]:

Edit Attachments

```
import requests
import json

# web service URL
access_token_url = 'https://api.ciscospark.com/v1/access_token'

# parameters for the API call
params = {
    'grant_type' : 'refresh_token',
    'client_id' : client_id,
    'client_secret' : client_secret,
    'refresh_token' : refresh_token
}

# POST
r = requests.post(access_token_url, json = params)

# the result is JSON
r = r.json()

print('JSON response:')
print(json.dumps(r, indent=4))

# check for errors
if r.get('errors'):
    error = r['errors'][0]
    print('Failed to get access token: {}'.format(error['description']))
else:
    access_token = r['access_token']
    refresh_token = r['refresh_token']

    print('\nAccess token: {}'.format(access_token))
    print('Valid for {} days'.format(round(r['expires_in'] / 60 / 60 / 24)))
    print('Refresh token: {}'.format(refresh_token))
    print('Valid for {} days'.format(round(r['refresh_token_expires_in'] / 60 / 60 / 24)))
```

Demo: Integration Authorization

```
# check for errors
if r.get('errors'):
    error = r['errors'][0]
    print('Failed to get access token: {}'.format(error['description']))
else:
    access_token = r['access_token']
    refresh_token = r['refresh_token']

    print('\nAccess token: {}'.format(access_token))
    print('Valid for {} days'.format(round(r['expires_in'] / 60 / 60 / 24)))
    print('Refresh token: {}'.format(refresh_token))
    print('Valid for {} days'.format(round(r['refresh_token_expires_in'] / 60 / 60 / 24)))
```

```
JSON response:
{
    "refresh_token": "OGY2M                zYyLTg0NWYtMTBhOTcwM2QxN2M2YWIyZGNiMmUtNmI5",
    "expires_in": 1209599,
    "refresh_token_expires_in": 7702526,
    "access_token": "Zjc0N               WU3LTk0ZjQtMGM2NWJmNjViMzQ5OGViZTM4NWUtNDRm"
}

Access token: Zjc0NjFhM                        GM2NWJmNjViMzQ5OGViZTM4NWUtNDRm
Valid for 14 days
Refresh token: OGY2M_    _ _  _         _  _   YtMTBhOTcwM2QxN2M2YWIyZGNiMmUtNmI5
Valid for 89 days
```

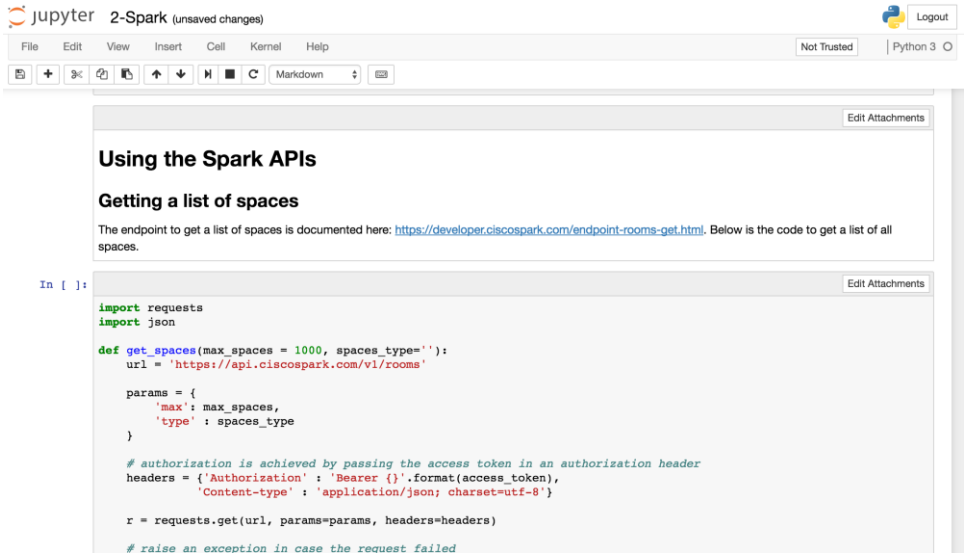
Deploying an Integration

- Register Integration at https://developer.cisco.com
- Redirect URL is part of the registration
- Redirect URL needs to be static and publicly available
- If deploying in the DMZ is not an option:
  - Paid Ngrok offering supports custom subdomains (https://example.ngrok.io)
  - .. and End-To-End TLS tunnels (use your own domains and certificates)
  - InfoSec probably doesn't like that either?
- Preferred: deploy on public hosting service
- .. but what if your service needs access to an internal backend?

Getting to Code

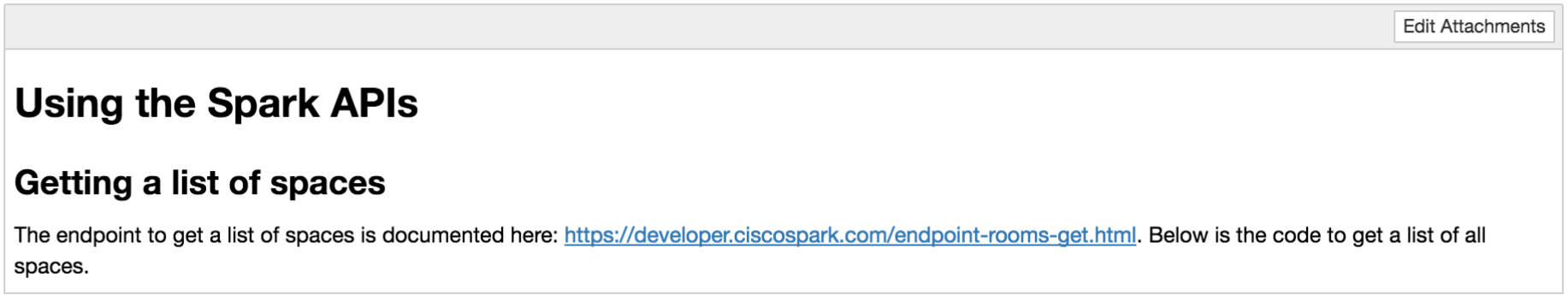
Demo: Spark Examples

- View the static notebooks at https://github.com/jeokrohn/brkcol2175clus2017/tree/master/Notebook

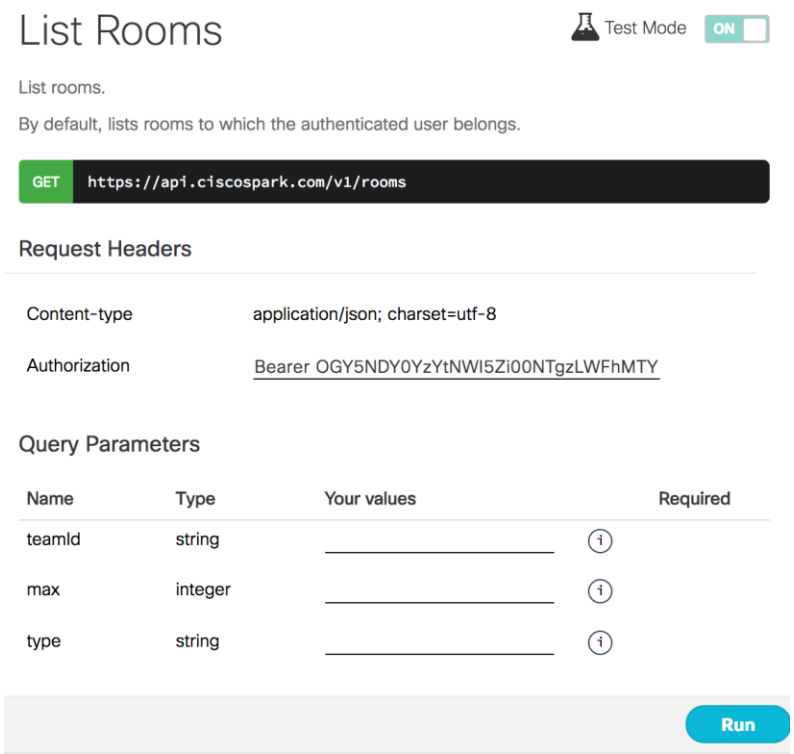


https://goo.gl/MbFC9k

Demo: Spark Examples



Demo: Spark Examples



Demo: Spark Examples

In [ ]:

Edit Attachments

```
import requests
import json

def get_spaces(max_spaces = 1000, spaces_type=''):
    url = 'https://api.ciscospark.com/v1/rooms'

    params = {
        'max': max_spaces,
        'type' : spaces_type
    }

    # authorization is achieved by passing the access token in an authorization header
    headers = {'Authorization' : 'Bearer {}'.format(access_token),
               'Content-type' : 'application/json; charset=utf-8'}

    r = requests.get(url, params=params, headers=headers)

    # raise an exception in case the request failed
    r.raise_for_status()

    # use the json() methof of the response object to get the response
    r = r.json()

    # the spaces are returned in the 'items' array of the response
    return r['items']
```

Demo: Spark Examples

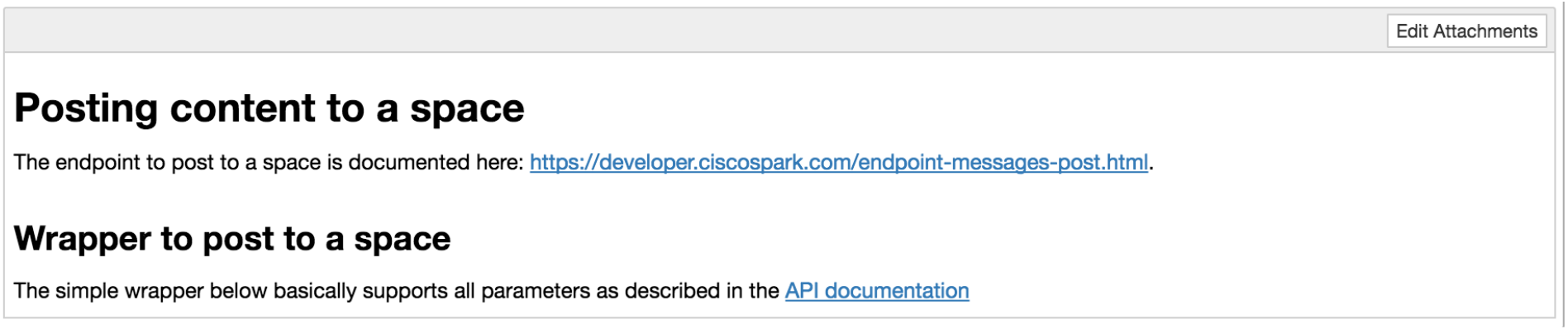
In [14]:

Edit Attachments

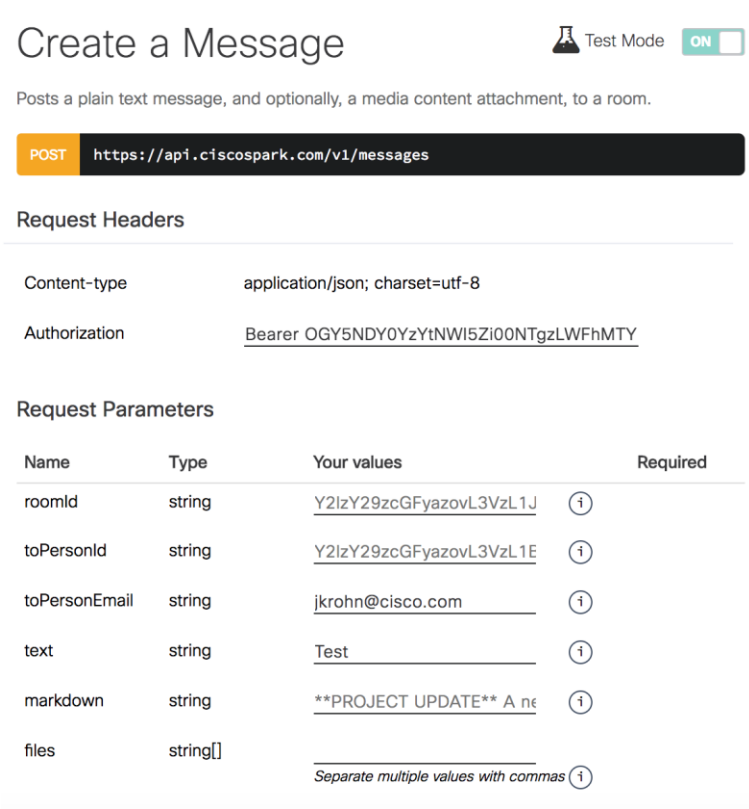
```
# get all spaces using the above function
spaces = get_spaces(max_spaces = 1000, spaces_type = 'group')

# create a sorted list of space titles
titles = [s['title'] for s in spaces]
titles.sort()

print('\n'.join(titles))
```

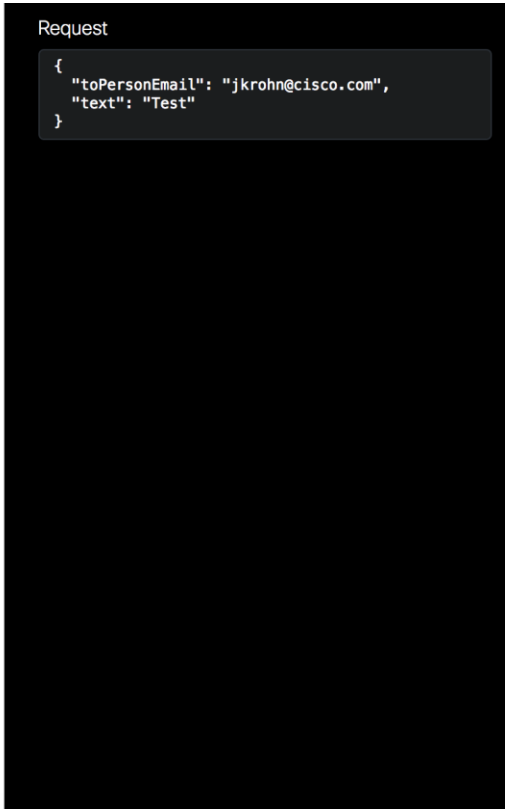
```
 Identity Lab PVT/SEVT 2016
 TME Demo Room
"Cloud Collaboration" Technical Training Collateral Planning
#spark4dev - Public Support for Cisco Spark API
(int)      - Spark ORG - get them rollin
11.5 GTM
2017-01-31 L2SIP, PSTN-INC000000016340
4:15 TODAY: be in Main Keynote for a thank you
ACE Collaboration
ACE leveraging Enterprise Trunk and conversion of Cisco alpha to Spark Call Ent PT with ByoPSTN
    - Collaboration Internal
APJ Partner Webinar
ATCG ( Acano Tactical Control Group) Thread 2.0 #community
ATS: PMP-SP Onboarding
Abdul Rahman
Acano for pa talk at CL
Adam Kalsey Bot Academy
Alan Glowacki
Alan, Tony, Hannes with Translators
```

Demo: Spark Examples



Demo: Spark Examples





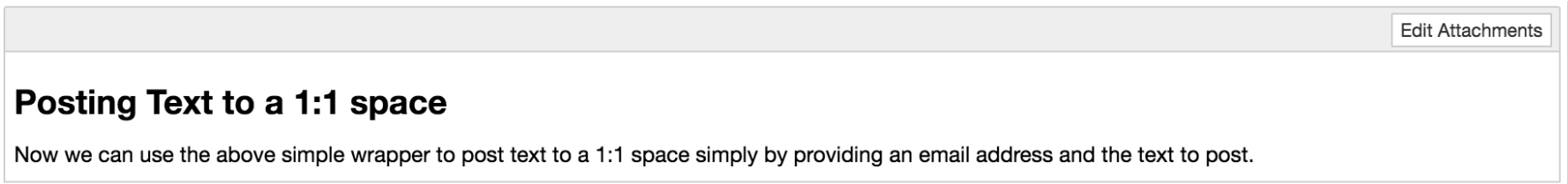
Demo: Spark Examples

In [ ]:

Edit Attachments

```
def post_message(roomId = None, toPersonId = None, toPersonEmail = None, text = None, markdown = None, files = None):
    params = {}

    # three different options to define destination
    if roomId is not None:
        params['roomId'] = roomId
    elif toPersonId is not None:
        params['toPersonId'] = toPersonId
    else:
        params['toPersonEmail'] = toPersonEmail

    # three different options to define the content
    if text is not None:
        params['text'] = text
    if markdown is not None:
        params['markdown'] = markdown
    if files is not None:
        params['files'] = files

    headers = {
        'Authorization' : 'Bearer {}'.format(access_token),
        'Content-Type' : 'application/json; charset=utf-8'
    }

    url = 'https://api.ciscospark.com/v1/messages'

    # the endpoint requires a POST and the parameters are passed as JSON in the body
    r = requests.post(url, json=params, headers=headers)

    return r.json()
```

Demo: Spark Examples



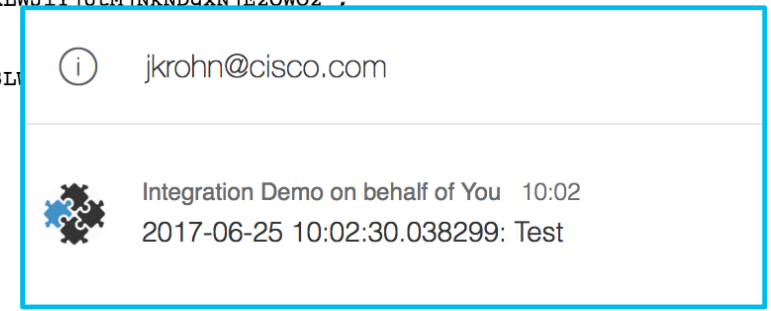
Demo: Spark Examples

In [18]:

Edit Attachments

```
import datetime

r = post_message(toPersonEmail = '        @gmail.com', text = '{}: Test'.format(datetime.datetime.now()))
print(json.dumps(r, indent=4))
```

```
{
    "toPersonEmail": "        @gmail.com",
    "text": "2017-06-25 10:02:30.038299: Test",
    "personId": "Y2lzY2                                        5Mjg5LTQ2NTUtOWE5NC0wNDNiOWVjMTMyOTk",
    "roomType": "direct",
    "roomId": "Y2lzY29zcGFyazovL3VzL1JPT00vYTEyYzhjOTEtNDE0Yi0zNzAxLWJlYjUtMjNkNDgxNjEzOWOz",
    "created": "2017-06-25T17:02:31.595Z",
    "personEmail": "jkrohn@cisco.com",
    "id": "Y2lzY29zcGFyazovL3VzL01FU1NBR0UvMTNhNDM3YjAtNTljOC0xMWU3LW
}
```



Demo: Spark Examples

Edit Attachments

Posting text to a group space

Posting to a regular group space requires the ID of the space to post to. This ID can be achieved by looking at the list of all spaces and searching for the space with the desired title

In [19]:

Edit Attachments

```
target_space_title = 'Jupyter Test'

spaces = get_spaces(spaces_type='group')

# (s for s in spaces) defines a generator which will return all spaces one by one if next() is called for the generator
# (s for s in spaces if s['title'] == target_space_title) is a generator for all spaces where the title matches
# calling next() on that generator returns the 1st space which matches the criteria expressed by 'if' in the generator
target_space = next((s for s in spaces if s['title'].startswith(target_space_title)))

print('target_space: {}'.format(target_space))
target_space_id = target_space['id']

print('\nTarget space \'{}\' has id {}'.format(target_space_title, target_space_id))
```

```
target_space: {'isLocked': False, 'creatorId': 'Y2lzY29zcGFyazovL3VzL1BFT1BMRS8wNzhkOGVjMi05Mjg5LTQ2NTUtOWE5NC0wNDNiO
WVjMTMyOTk', 'id': 'Y2lzY29zcGFyazovL3VzL1JPT00vM2Q3MjE0ODAtNDA3NC0xMWU3LTk3MDgtMGRmOWNkNzFjNWQx', 'lastActivity': '2
017-06-20T16:37:26.593Z', 'created': '2017-05-24T11:29:24.936Z', 'type': 'group', 'title': 'Jupyter Test'}

Target space 'Jupyter Test' has id Y2lzY29zcGFyazovL3VzL1JPT00vM2Q3MjE0ODAtNDA3NC0xMWU3LTk3MDgtMGRmOWNkNzFjNWQx
```

Demo: Spark Examples

Edit Attachments

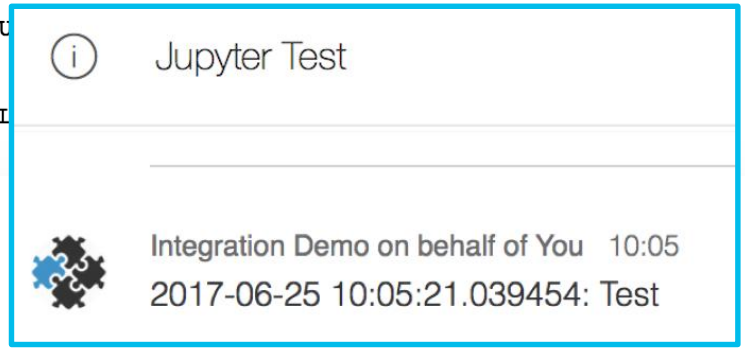
With that space ID we can now again post text using our wrapper.

In [20]:

Edit Attachments

```
import datetime

r = post_message(roomId = target_space_id, text = '{}: Test'.format(datetime.datetime.now()))
print(json.dumps(r, indent=4))
```

```
{
    "text": "2017-06-25 10:05:21.039454: Test",
    "id": "Y2lzY29zcGFyazovL3VzL01FU1NBR0UvN2E2MzUxNzAtNTljOC0xMWU3LTg1ZTUtMTU4MWEyMjJlNGEw",
    "roomType": "group",
    "roomId": "Y2lzY29zcGFyazovL3VzL1JPT00vM2Q3MjE0ODAtNDA3NC0xMWU
    "created": "2017-06-25T17:05:23.975Z",
    "personEmail": "jkrohn@cisco.com",
    "personId": "Y2lzY29zcGFyazovL3VzL1BFT1BMRS8wNzhkOGVjMi05Mjg5L
}
```



Demo: Spark Examples

Edit Attachments

Formatting messages

Markdown can be used to post rich text content. The documentation of the available markdown syntax can be found at developer.ciscospark.com

In [21]:

Edit Attachments

```
markdown = '''{}: Messages can contain [links](http://www.ciscolive.com/online).

And multiple lines of text.

**Bold** text is also possible.

Lists
- can
- contain
- multiple
- entries

Or
1. can
2. be
3. ordered
'''.format(datetime.datetime.now())
r = post_message(roomId = target_space_id, markdown = markdown)
print(json.dumps(r, indent=4))
```

Demo: Spark Examples


```
{
    "html": "<p>2017-06-25 10:07:57.253073: Messages can contain <a href=\"http://www.ciscolive.com/online\">links</a
>.</p><p>And multiple lines of text.</p><p><strong>Bold</strong> text is also possible.</p><p>Lists</p><ul><li>can</l
i><li>contain</li><li>multiple</li><li>entries</li></ul><p>Or</p><ol><li>can</li><li>be</li><li>ordered</li></ol>",
    "text": "2017-06-25 10:07:57.253073: Messages can contain links. And multiple lines of text. Bold text is also po
ssible. Lists can contain multiple entries Or can be ordered",
    "id": "Y2lzY29zcGFyazovL3VzL01FU1NBR0UvZDcwNWMzZTAtNTljOC0xMWU3LWE0NmEtZjlmNzllYzg5YzU3",
    "roomType": "group",
    "roomId": "Y2lzY29zcGFyazovL3VzL1JPT00vM2Q3MjE0ODAtNDA3NC0xMWU3LTk3MDgtMGRmOWNkNzFjNWQx",
    "created": "2017-06-25T17:07:59.390Z",
    "personEmail": "jkrohn@cisco.com",
    "markdown": "2017-06-25 10:07:57.253073: Messages can contain [links](http://www.ciscolive.com/online).\n\nAnd mu
ltiple lines of text.\n\n**Bold** text is also possible.\n\nLists\n- can\n- contain\n- multiple\n- entries\n\nOr\n1.
 can\n2. be\n3. ordered\n",
    "personId": "Y2lzY29zcGFyazovL3VzL1BFT1BMRS8wNzhkOGVjMi05Mjg5LTQ2NTUtOWE5NC0wNDNiOWVjMTMyOTk"
}
```

Demo: Spark Examples

Integration Demo on behalf of You 10:07

2017-06-25 10:07:57.253073: Messages can contain links.

And multiple lines of text.

Bold text is also possible.

Lists

- can
- contain
- multiple
- entries

Or

1. can
2. be
3. ordered

Demo: Spark Examples

Edit Attachments

Posting Attachments

Public Files

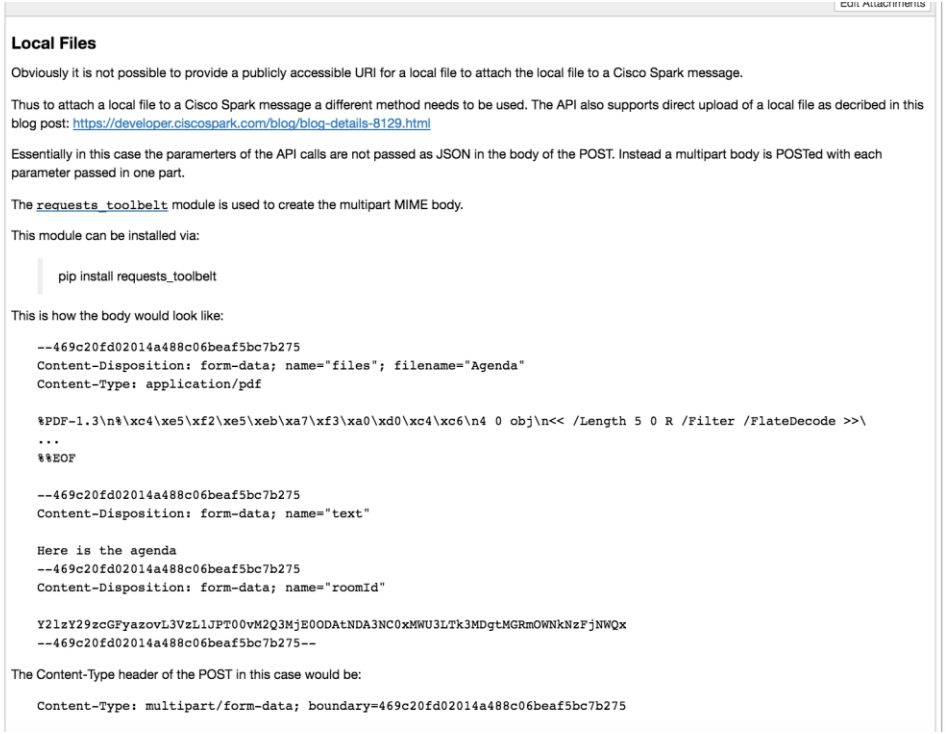
The API allows to pass a publich URL of a file to be posted to Cisco Spark as an attachment.

In [*]:

Edit Attachments

```
# list of publicly accessible URLs of some traffic cams in Germany
urls = ['http://autobahn-rlp.de/syncdata/cam/380/thumb_640x480.jpg',
        'http://autobahn-rlp.de/syncdata/cam/385/thumb_640x480.jpg',
        'http://autobahn-rlp.de/syncdata/cam/165/thumb_640x480.jpg']

text = 'Traffic in Germany'

r = post_message(roomId = target_space_id, text=text)

# now for each traffic cam
for url in urls:
    # post a message with one attachment defined by that public URL
    r = post_message(roomId = target_space_id, files=[url])
```

Demo: Spark Examples



Local Files

Obviously it is not possible to provide a publicly accessible URI for a local file to attach the local file to a Cisco Spark message.

Thus to attach a local file to a Cisco Spark message a different method needs to be used. The API also supports direct upload of a local file as decribed in this blog post: https://developer.ciscospark.com/blog/blog-details-8129.html

Essentially in this case the paramerters of the API calls are not passed as JSON in the body of the POST. Instead a multipart body is POSTed with each parameter passed in one part.

The `requests_toolbelt` module is used to create the multipart MIME body.

This module can be installed via:

```
pip install requests_toolbelt
```

This is how the body would look like:

```
--469c20fd02014a488c06beaf5bc7b275
Content-Disposition: form-data; name="files"; filename="Agenda"
Content-Type: application/pdf

%PDF-1.3\n%\xc4\xe5\xf2\xe5\xeb\xa7\xf3\xa0\xd0\xc4\xc6\n4 0 obj\n<< /Length 5 0 R /Filter /FlateDecode >>\
...
%%EOF

--469c20fd02014a488c06beaf5bc7b275
Content-Disposition: form-data; name="text"

Here is the agenda
--469c20fd02014a488c06beaf5bc7b275
Content-Disposition: form-data; name="roomId"

Y2lzY29zcGFyazovL3VzL1JPT00vM2Q3MjE0ODAtNDA3NC0xMWU3LTk3MDgtMGRmOWNkNzFjNWQx
--469c20fd02014a488c06beaf5bc7b275--
```

The Content-Type header of the POST in this case would be:

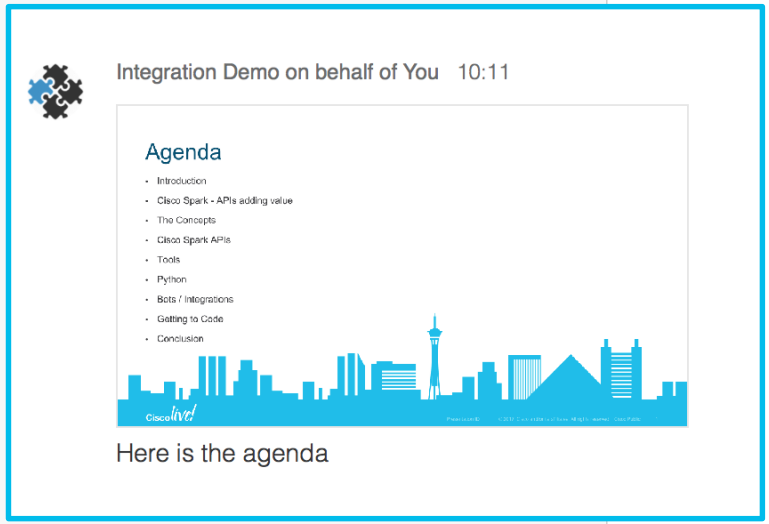
```
Content-Type: multipart/form-data; boundary=469c20fd02014a488c06beaf5bc7b275
```

Demo: Spark Examples

In [ ]:

Edit Attachments

```
import requests_toolbelt
import requests

file_path = './BRKCOL-2175_Agenda.pdf'
file_type = 'application/pdf'
endpoint = "https://api.ciscospark.com/v1/messages"

with open(file_path, 'rb') as f:
    # prepare the multipart body
    data = {'roomId': target_space_id,
            'text': 'Here is the agenda',
            'files': ('Agenda', f, file_type)
           }
    multi_part = requests_toolbelt.MultipartEncoder(fields=data)

    headers = {'Content-Type': multi_part.content_type,
            'Authorization': 'Bearer {}'.format(access_token)}

    r = requests.post(endpoint, data=multi_part, headers=headers)

r.raise_for_status()

print(json.dumps(r.json(), indent=4))

print('\nThe request has been sent with\n\'Content-Type: {}\''.format(r.request.headers['Content-Type']))
```



Demo: Spark Examples

Edit Attachments

Simplify Spark API usage by using an API wrapper

While creating wrappers for all methods offered by the Cisco Spark API is pretty straight forward (see the `post_message()` and `get_spaces()` examples above) it's even easier (and more efficient) to use a readily available API wrapper which supports all operations offered by the Cisco Spark API.

One example is the ciscosparkapi API wrapper. It can be installed via:

```
pip install ciscosparkapi
```

Documentation is hosted at http://ciscosparkapi.readthedocs.io/en/latest/index.html

Edit Attachments

Create the API object

The **`CiscoSparkAPI`** object is instantiated with the existing OAuth access token which will then be used for all API calls by the wrapper.

Edit Attachments

```
import ciscosparkapi

api = ciscosparkapi.CiscoSparkAPI(access_token=access_token)
```

Demo: Spark Examples

Edit Attachments

Find a space

The target space for our tests again can be found by going through the list of spaces and looking for the right title. The API wrapper instead of returning JSON (or Python dictionaries as in our examples above) returns objects. For example the elements returned by the generator created by the `rooms.list()` call are Room objects so that we have to look at the `.title` attribute instead of looking at the `['title']` member of a dictionary.

In [25]:

Edit Attachments

```
import ciscosparkapi

api = ciscosparkapi.CiscoSparkAPI(access_token=access_token)

spaces = api.rooms.list(max=500, type='group')
target_space = next((s for s in spaces if s.title.startswith(target_space_title)))
target_space
```

Out[25]:

```
Room({"isLocked": false, "creatorId": "Y2lzY29zcGFyazovL3VzL1BFT1BMRS8wNzhkOGVjMi05Mjg5LTQ2NTUtOWE5NC0wNDNiOWVjMTMyOTk", "id": "Y2lzY29zcGFyazovL3VzL1JPT00vM2Q3MjE0ODAtNDA3NC0xMWU3LTk3MDgtMGRmOWNkNzFjNWQx", "lastActivity": "2017-06-25T17:11:02.025Z", "created": "2017-05-24T11:29:24.936Z", "type": "group", "title": "Jupyter Test"})
```

Demo: Spark Examples

Iterate through messages

The Wrapper also allows to easily iterate through all messages posted in a given space.

In [26]:

Edit Attachments

```
import ciscosparkapi

api = ciscosparkapi.CiscoSparkAPI(access_token=access_token)

messages = api.messages.list(roomId=target_space.id, max=1000)
for m in messages:
    print(m)
```

```
Message:
{
  "files": [
    "https://api.ciscospark.com/v1/contents/Y2lzY29zcGFyazovL3VzL0NPTlRFTlQvNDNlMWE3OTAtNTljOS0xMWU3LTg1ZTUtMTU4MWE
yMjJlNGEwLzA"
  ],
  "text": "Here is the agenda",
  "personId": "Y2lzY29zcGFyazovL3VzL1BFT1BMRS8wNzhkOGVjMi05Mjg5LTQ2NTUtOWE5NC0wNDNiOWVjMTMyOTk",
  "roomType": "group",
  "roomId": "Y2lzY29zcGFyazovL3VzL1JPT00vM2Q3MjE0ODAtNDA3NC0xMWU3LTk3MDgtMGRmOWNkNzFjNWQx",
  "created": "2017-06-25T17:11:02.025Z",
  "personEmail": "jkrohn@cisco.com",
  "id": "Y2lzY29zcGFyazovL3VzL01FU1NBR0UvNDNlMWE3OTAtNTljOS0xMWU3LTg1ZTUtMTU4MWEyMjJlNGEw"
}
Message:
{
  "files": [
    "https://api.ciscospark.com/v1/contents/Y2lzY29zcGFyazovL3VzL0NPTlRFTlQvMDJlYWE0ODAtNTljOS0xMWU3LWFhMmEtNTkwODI
```

Demo: Spark Examples

Edit Attachments

Filtering messages (messages posted by me)

Using the same mechanisms as above we can easily filter for messages posted by ourself by looking at the `personId` of the messages in the space.

In [27]:

Edit Attachments

```
import ciscosparkapi

api = ciscosparkapi.CiscoSparkAPI(access_token=access_token)

me = api.people.me()

messages = api.messages.list(roomId=target_space.id, max=2000)

messages_posted_by_me = [m for m in messages if m.personId == me.id]

print('{} messages posted by me'.format(len(messages_posted_by_me)))

messages_posted_by_me
```

```
7 messages posted by me
```

Out[27]:

```
[Message({"files": ["https://api.ciscospark.com/v1/contents/Y2lzY29zcGFyazovL3VzL0NPTlRFTlQvNDNlMWE3OTAtNTljOS0xMWU3L
Tg1ZTUtMTU4MWEyMjJlNGEwLzA"], "text": "Here is the agenda", "personId": "Y2lzY29zcGFyazovL3VzL1BFT1BMRS8wNzhkOGVjMi05
Mjg5LTQ2NTUtOWE5NC0wNDNiOWVjMTMyOTk", "roomType": "group", "roomId": "Y2lzY29zcGFyazovL3VzL1JPT00vM2Q3MjE0ODAtNDA3NC0
xMWU3LTk3MDgtMGRmOWNkNzFjNWQx", "created": "2017-06-25T17:11:02.025Z", "personEmail": "jkrohn@cisco.com", "id": "Y2lz
Y29zcGFyazovL3VzL01FU1NBR0UvNDNlMWE3OTAtNTljOS0xMWU3LTg1ZTUtMTU4MWEyMjJlNGEw"}),
 Message({"files": ["https://api.ciscospark.com/v1/contents/Y2lzY29zcGFyazovL3VzL0NPTlRFTlQvMDJlYWE0ODAtNTljOS0xMWU3L
WFhMmEtNTkwODIwYWIzNzVhLzA"], "roomId": "Y2lzY29zcGFyazovL3VzL1JPT00vM2Q3MjE0ODAtNDA3NC0xMWU3LTk3MDgtMGRmOWNkNzFjNWQ
x", "id": "Y2lzY29zcGFyazovL3VzL01FU1NBR0UvMDJlYWE0ODAtNTljOS0xMWU3LWFhMmEtNTkwODIwYWIzNzVh", "roomType": "group", "c
```

Demo: Spark Examples

Creating messages

The API wrapper also supports greating messages. When creating messages with attachments (`files` parameter is used) then the API makes sure to use multipart messages to upload if a local file is referenced in the `files` argument.

In [*]:

Edit Attachments

```
import ciscosparkapi

api = ciscosparkapi.CiscoSparkAPI(access_token=access_token)

api.messages.create(roomId=target_space.id, text='Posted via the ciscosparkapi API')

api.messages.create(roomId=target_space.id, text='Posted via the ciscosparkapi API', markdown='Posted via the **`ciscosp

files = ['http://autobahn-rlp.de/syncdata/cam/380/thumb_640x480.jpg',
         'http://autobahn-rlp.de/syncdata/cam/385/thumb_640x480.jpg',
         'http://autobahn-rlp.de/syncdata/cam/165/thumb_640x480.jpg',
         './BRKCOL-2175_Agenda.pdf']

api.messages.create(roomId=target_space.id, text='Some attachments')
# need to post the attachments individually as the Cisco Spark API currently only supports one attachment at a time.
for file in files:
    api.messages.create(roomId=target_space.id, files=[file])
```

Demo: Spark Examples

Looking at memberships

The memberships API app allows to read, or update Cisco Spark space memberships.

In [29]:

Edit Attachments

```
import ciscosparkapi
import base64

api = ciscosparkapi.CiscoSparkAPI(access_token=access_token)

members = api.memberships.list(roomId=target_space.id)
for m in members:
    print(m)
```

```
Membership:
{
  "isModerator": false,
  "roomId": "Y2lzY29zcGFyazovL3VzL1JPT00vM2Q3MjE0ODAtNDA3NC0xMWU3LTk3MDgtMGRmOWNkNzFjNWQx",
  "id": "Y2lzY29zcGFyazovL3VzL01FTUJFUlNISVAvMDc4ZDhlYzItOTI4OS00NjU1LTlhOTQtMDQzYjllYzEzMjk5OjNkNzIxNDgwLTQwNzQtMTFl
Ny05NzA4LTBkZjljZDcxYzVkMQ",
  "personDisplayName": "Johannes Krohn",
  "personOrgId": "Y2lzY29zcGFyazovL3VzL09SR0FOSVpBVElPTi8xZWI2NWZkZi05NjQzLTQxN2YtOTk3NC1hZDcyY2FlMGUxMGY",
  "created": "2017-05-24T11:29:25.509Z",
  "personEmail": "jkrohn@cisco.com",
  "personId": "Y2lzY29zcGFyazovL3VzL1BFT1BMRS8wNzhkOGVjMi05Mjg5LTQ2NTUtOWE5NC0wNDNiOWVjMTMyOTk",
  "isMonitor": false
}
Membership:
{
  "isModerator": false,
```

Demo: Spark Examples

Bulk deleting messages

With the list of messages posted by me we can easily delete all messages created by me

!!!BE CAREFUL WITH THIS CODE!!!

!!!MAKE SURE (TRIPPLE CHECK) TO USE A TEST SPACE ONLY!!!

In [30]:

Edit Attachments

```
import ciscosparkapi

api = ciscosparkapi.CiscoSparkAPI(access_token=access_token)

me = api.people.me()

# list of message IDs posted by me in the above space
my_message_ids = [m.id for m in api.messages.list(roomId=target_space.id) if m.personId == me.id]

print('Deleting {} messsages'.format(len(my_message_ids)))

# enumerate() returns an iterator which returns tuples (index, element of iterable passed to enumerate)
# reversed() created an iterator which returs elements of the passed list in reversed order
# using reversed() as we want to delete the messages in chronological order: oldest 1st
for i, id in enumerate(reversed(my_message_ids)):
    print('Deleting message {}/{}'.format(i+1, id))
    api.messages.delete(id)
```

```
Deleting 14 messsages
Deleting message 1/Y2lzY29zcGFyazovL3VzL01FU1NBR0UvN2E2MzUxNzAtNTljOC0xMWU3LTg1ZTUtMTU4MWEyMjJlNGEw
Deleting message 2/Y2lzY29zcGFyazovL3VzL01FU1NBR0UvZDcwNWMzZTAtNTljOC0xMWU3LWE0NmEtZjlmNzllYzg5YzU3
```

Building a Basic Bot using Python

- Start a local web server on an available port
- Need to start an Ngrok process for redirection of a public URI to our local web server
- Register a Webhook for the Bot to subscribe to updates
- Service all incomings POSTs
- Running Code in the Jupiter Notebook of this session!
- Demo Video available in GitHub repository

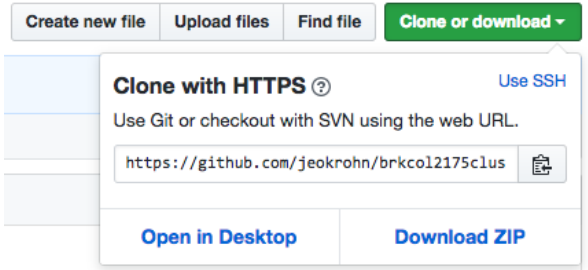
Additional Material

Docker

- “Open source container software platform that packages applications in containers”
- The live Notebooks run in a Docker container
- Docker needs to be installed on the local machine.
- Installation:
  - Mac: https://www.docker.com/docker-mac
  - Windows: https://www.docker.com/docker-windows
- Free Community Edition is sufficient to run the Notebook container

Git Repository

- Repository available at: https://github.com/jeokrohn/brkcol2175clus2017.git
- Contents:
  - Jupyter notebooks (static)
  - PDF of session presentation
  - Demo videos



https://goo.gl/iLiwcm

Jupyter Notebook

- “The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and explanatory text”
- Used in this session only to demonstrate live Python Code
- Notebooks used in this session are available at https://github.com/jeokrohn/brkcol2175clus2017.git
- The GitHub repository allows you to build a Docker image to run a live notebook
- Documentation at GitHub

https://goo.gl/iLiwcm

Live Jupyter Demo Notebooks

- You can build your own Docker image with a Jupyter server and the session Notebooks: see Readme on GitHub
- .. or use the prepared image

```
docker run -it --rm --name jupyter -p 8888:8888 jeokrohn/brkcol2175clus2017
```

- Point your browser to http://localhost:8888 to access the notebooks
- The password to access the server is: ‘brkcol2175’
- If port 8888 on your local machine is not available then use different port mappings (for example –p 8889:8888 to use local port 8889)

Demo Videos

1. Developer.cisco.com
2. Python basics
3. OAUTH flow using Python
4. Getting to Code (Using the Spark APIs from Python)
5. Building a Bot

https://goo.gl/csvqYr

- All videos are published in the GitHub repository (see before)
- .. and on Box: https://cisco.box.com/v/brkcol2175clus2017

Conclusion

Follow-Up

- Review Demo Videos
- Review examples code in Jupyter notebooks
  - Live: run the docker image (see Appendix)
  - Static: browse notebooks on GitHub (see Appendix)
- Play with and extend code in the demo notebooks
- Use the Cisco Spark Space for follow-up conversation
- Start your own Python project
- Ideas:
  - Find “old” spaces
  - Download attachments from spaces
  - …

Conclusion

- Python is great!
- Coding is fun!
- Cisco Spark APIs allow to extend the Cisco Spark eco-system

Cisco Spark

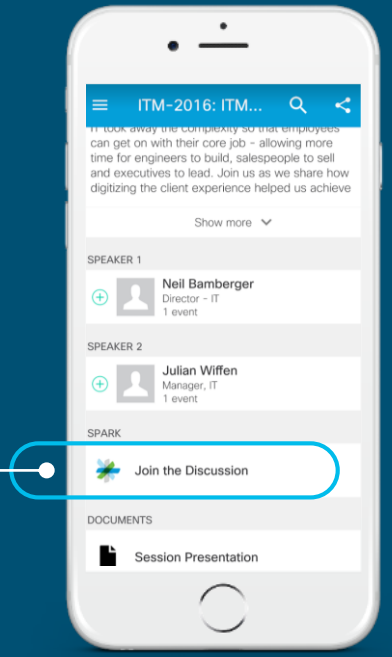
Questions?

Use Cisco Spark to communicate with the speaker after the session

How

1. Find this session in the Cisco Live Mobile App
2. Click “Join the Discussion”
3. Install Spark or go directly to the space
4. Enter messages/questions in the space

Cisco Spark spaces will be available until July 3, 2017.



cs.co/ciscolivebot#BRKCOL-2175

Complete Your Online Session Evaluation

- Give us your feedback to be entered into a Daily Survey Drawing. A daily winner will receive a $750 gift card.
- Complete your session surveys through the Cisco Live mobile app or on www.CiscoLive.com/us.

Don’t forget: Cisco Live sessions will be available for viewing on demand after the event at www.CiscoLive.com/Online.

Ciscolive!

Continue Your Education

- Demos in the Cisco campus
- Walk-in Self-Paced Labs
- Lunch & Learn
- Meet the Engineer 1:1 meetings
- Related sessions

Thank you


cisco
You're
Cisco live!

Collaboration Cisco Education Offerings

Course	Description	Cisco Certification
CCIE Collaboration Advanced Workshop (CIEC)	Gain expert-level skills to integrate, configure, and troubleshoot complex collaboration networks	CCIE® Collaboration
Implementing Cisco Collaboration Applications (CAPPS)	Understand how to implement the full suite of Cisco collaboration applications including Jabber, Cisco Unified IM and Presence, and Cisco Unity Connection.	CCNP® Collaboration
Implementing Cisco IP Telephony and Video Part 1 (CIPTV1) Implementing Cisco IP Telephony and Video Part 2 (CIPTV2) Troubleshooting Cisco IP Telephony and Video (CTCOLLAB)	Learn how to implement Cisco Unified Communications Manager, CUBE, and audio and videoconferences in a single-site voice and video network. Obtain the skills to implement Cisco Unified Communications Manager in a modern, multisite collaboration environment. Troubleshoot complex integrated voice and video infrastructures	CCNP® Collaboration
Implementing Cisco Collaboration Devices (CICD) Implementing Cisco Video Network Devices (CIVND)	Acquire a basic understanding of collaboration technologies like Cisco Call Manager and Cisco Unified Communications Manager. Learn how to evaluate requirements for video deployments, and implement Cisco Collaboration endpoints in converged Cisco infrastructures.	CCNA® Collaboration

For more details, please visit: http://learningnetwork.cisco.com
Questions? Visit the Learning@Cisco Booth