CISCO

You make **possible**

Extending the Collaboration Eco-System using Webex Teams APIs for Non-Developers

Johannes Krohn
Technical Marketing Engineer

BRKCOL-2175

Barcelona | January 27-31, 2020

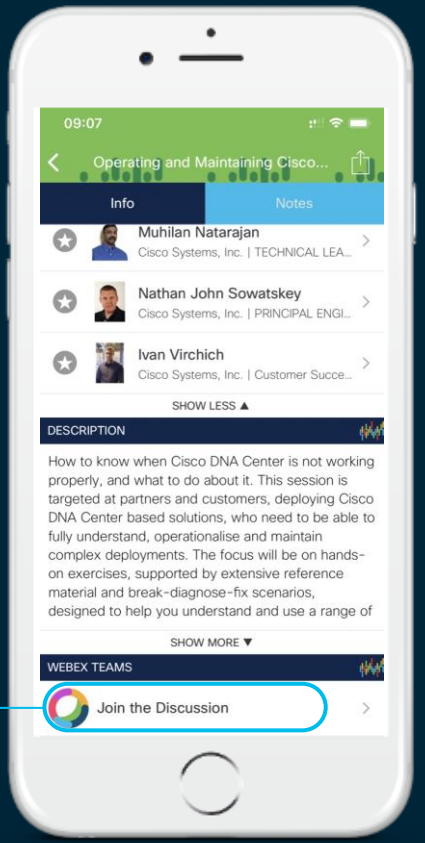
Cisco Webex Teams

Questions?

Use Cisco Webex Teams to chat with the speaker after the session

How

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



Abstract

Extensibility through open APIs is one of the key characteristics of Webex Teams. 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 Webex Teams space for this session.

Agenda

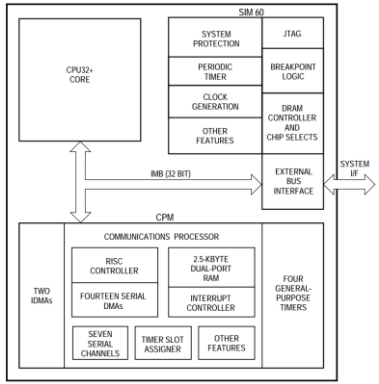
- Introduction
- Webex Teams – APIs adding value
- The Concepts
- Webex Teams APIs
- Tools
- Python
- Bots / Integrations
- Getting to Code
- Conclusion

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

python

REST JSON

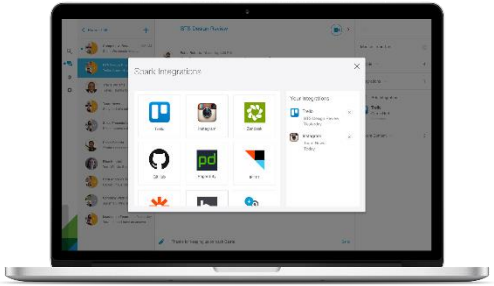
Webex Teams – APIs Adding Value

Make Webex Teams the Place for All Your Work

Native Integrations

Easy to configure native integrations in the Webex Teams app

pagerduty GitHub Instagram Trello zendesk



Integration Services

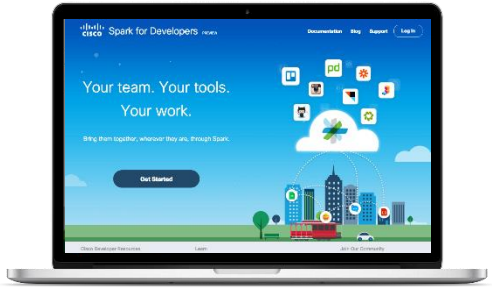
Integrate with other apps in seconds to automate tasks and make your life more efficient

Built.io Zapier IFTTT

Webex for Developers

Create custom integrations using the rich Webex Teams APIs

developer.webex.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

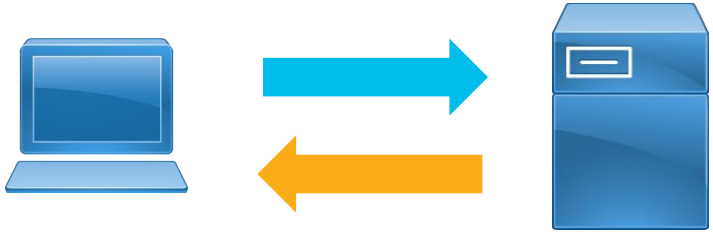
JSON – JavaScript Object Notation

- 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



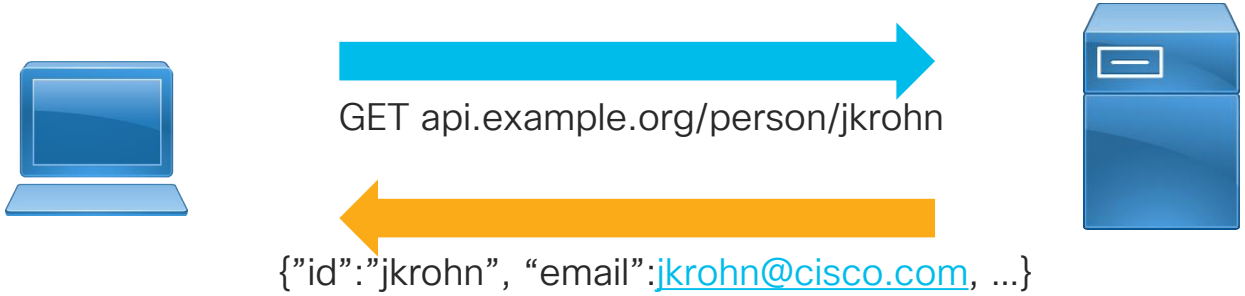
REST Methods

- 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

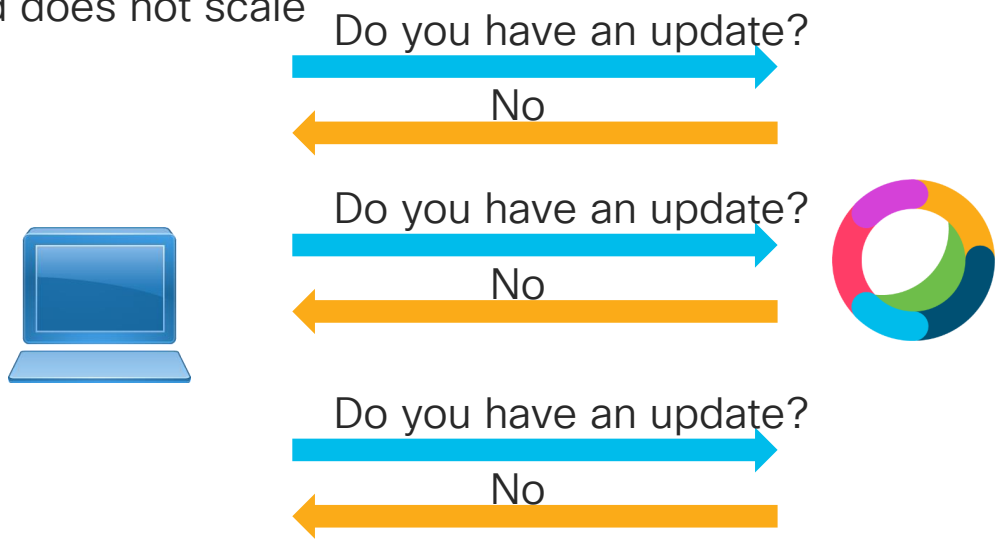


Webhooks

Webhooks – Problem Statement

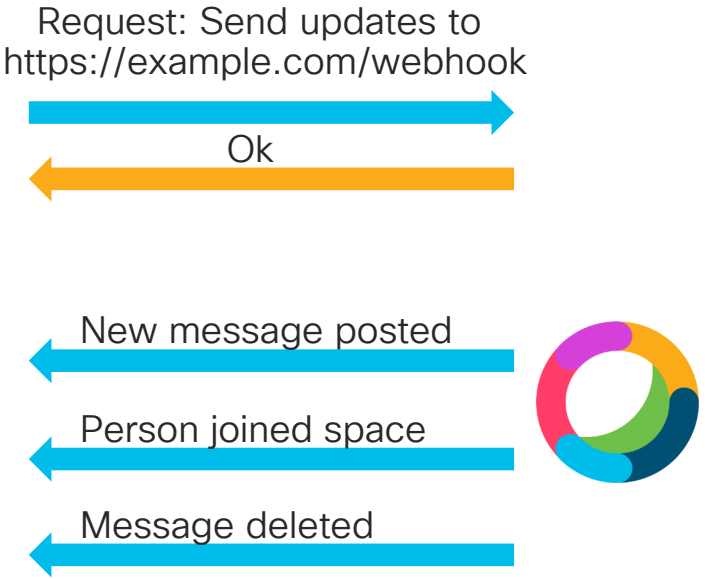
- Polling for events is inefficient and does not scale
- Too many instances polling
- Too many event types to poll for

  → not really a (scalable) option



Webhooks – Concept

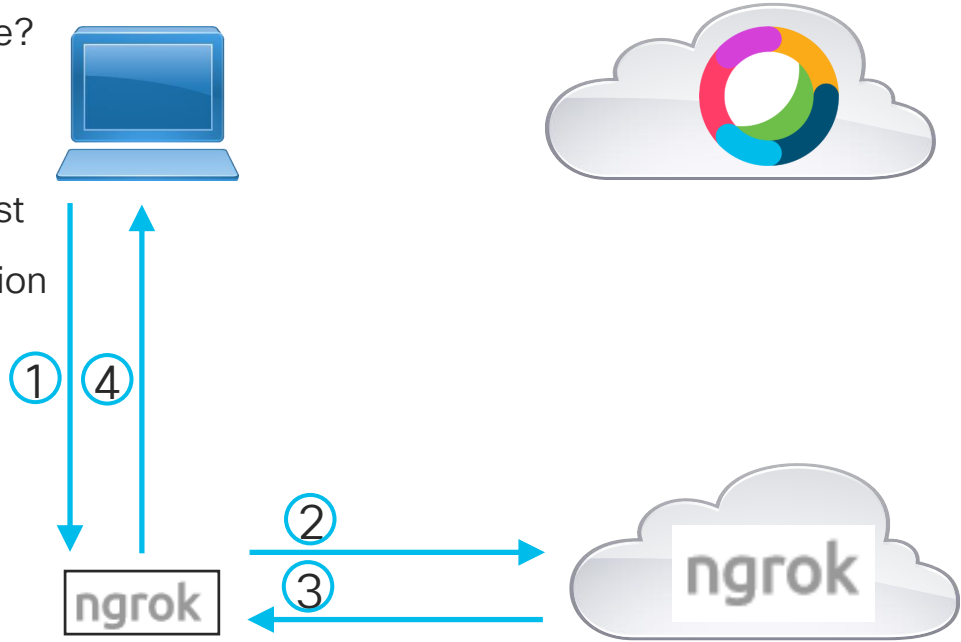
- Ask for notifications
- Register Webhook
  - HTTP callback
- Web service “calls” Webhook
  - POST to registered URL
- Publish/Subscribe instead of Polling
- Requires public URL for callbacks



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

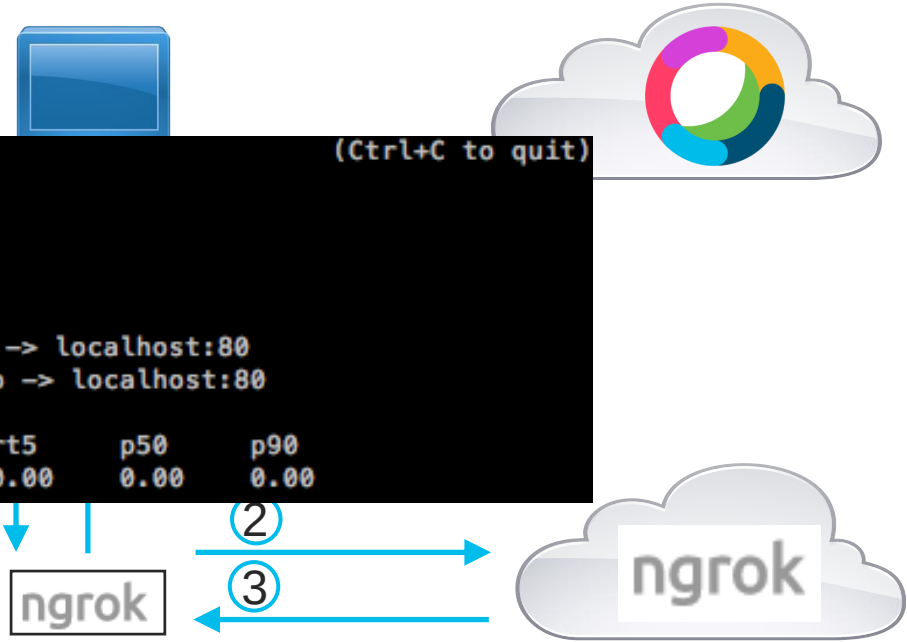
1. Start ngrok client
2. Create persistent connection
3. Obtain public URL
4. Report public URL



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

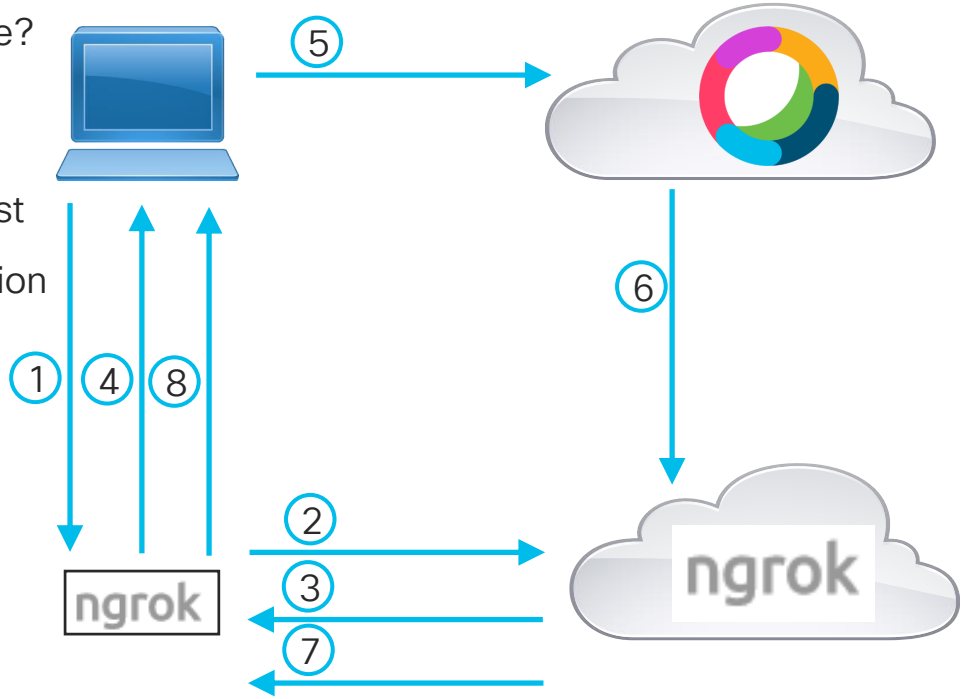
3 Obtain public URL

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



Webex Teams APIs

Webex Teams APIs

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

API Reference

Events

Licenses

Memberships

Messages

Organizations

People

Resource Group Memberships

Resource Groups

Roles

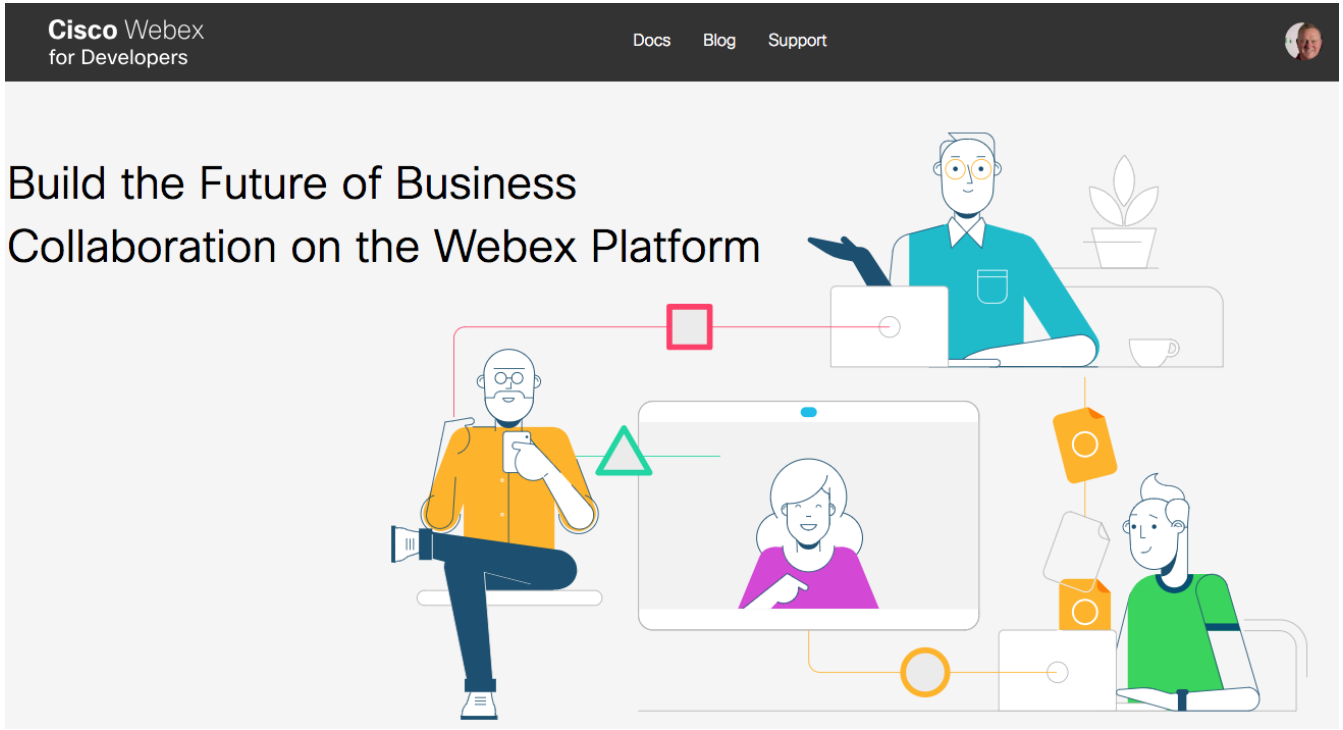
Rooms

Team Memberships

Teams

Webhooks

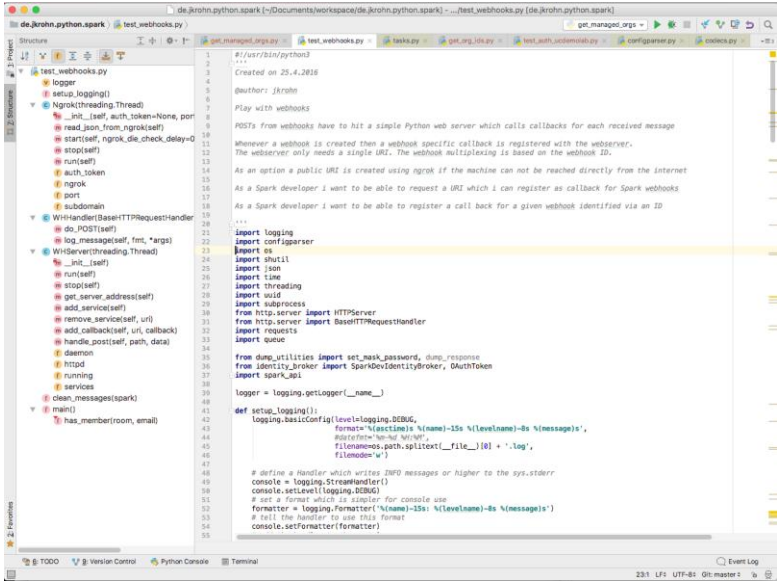
Demo: Webex Teams API Documentation



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

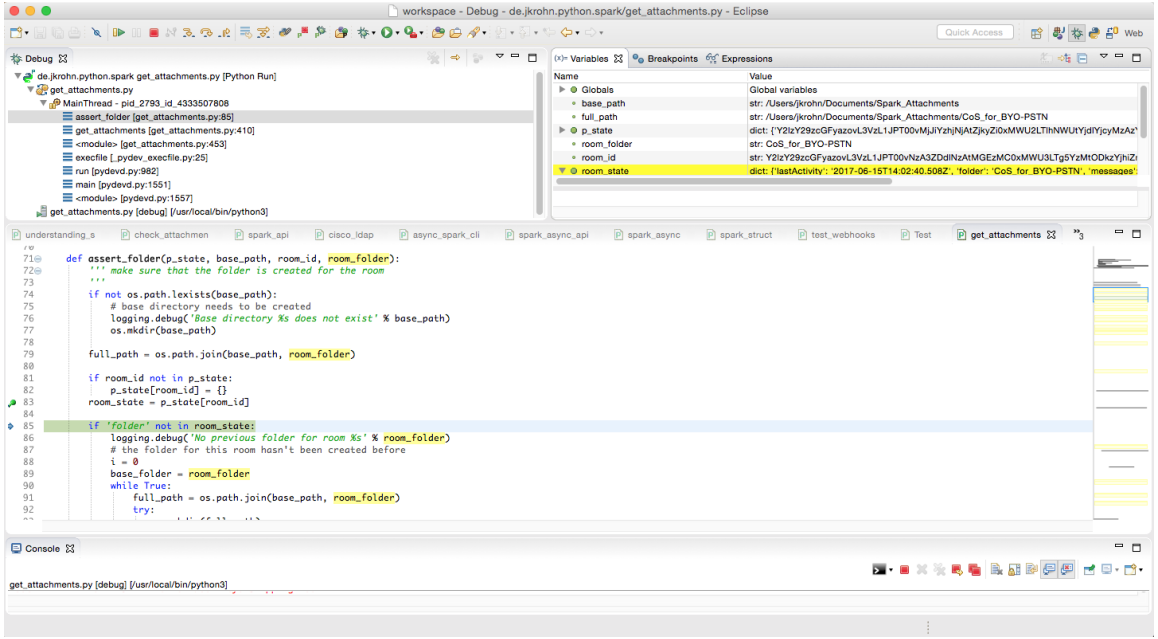
- What Do you prefer?
- Or this?

```
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

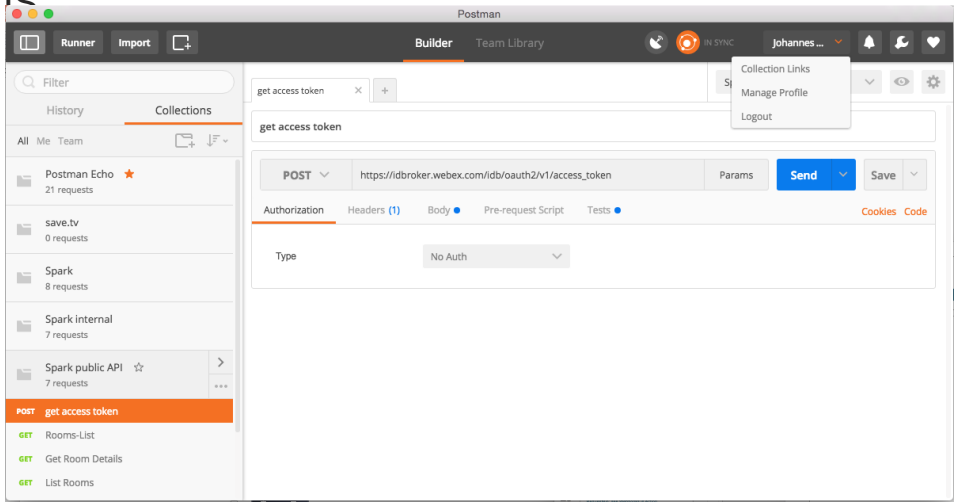


IDEs for Python

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

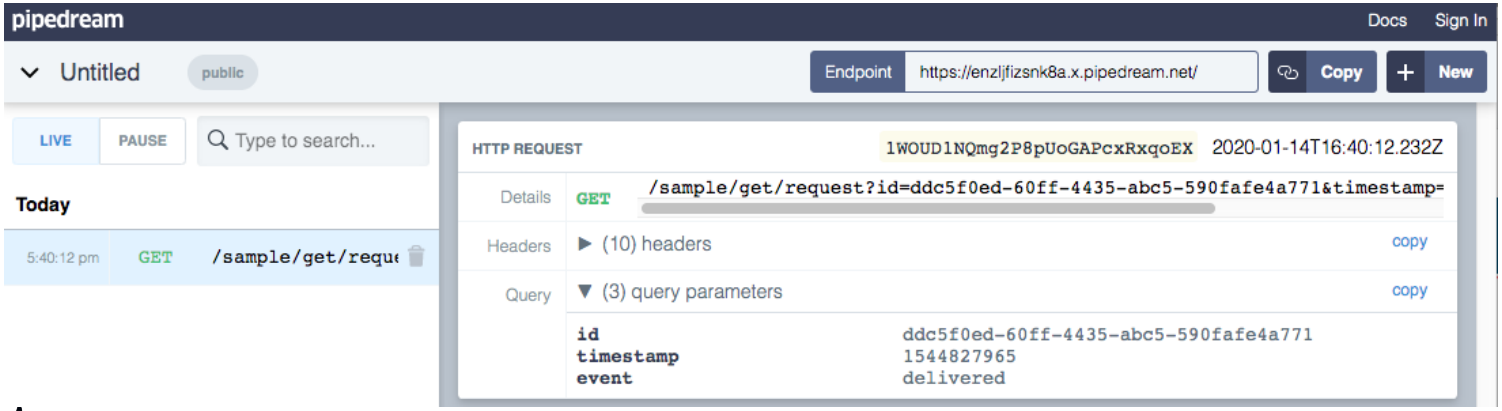
Postman: Test APIs

- 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/
- Postman collection for Webex Teams: https://github.com/CiscoDevNet/postman-webex



RequestBin: See Webhooks in Action

- Free service: https://requestbin.com
- Creates unique URL
- Use case: Webex Teams webhook pointing to Requestbin 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
  - 2019: Python 3.8 released
- New feature development only in 3.x
- Recommendation: start with latest Python release (3.8)

Learning Python

- Beginner's Guide: https://wiki.python.org/moin/BeginnersGuide
- The Hitchhiker's Guide to Python: https://docs.python-guide.org/
- Online Courses
  - https://www.edx.org/learn/python
  - https://www.coursera.org/specializations/python
  - https://www.codecademy.com/learn/learn-python-3
- 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 3.7.5 (v3.7.5:5c02a39a0b, Oct 14 2019, 18:49:57)
[Clang 6.0 (clang-600.0.57)] 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"
}
>>>
```

Integrations vs Bots

Integration

Request permission (OAuth) to invoke Webex Teams APIs on behalf of another user.

Learn More

Create an Integration

Bot

Build intelligent chatbots that post content and respond to commands.

Learn More

Create a Bot

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 Webex Teams messages they are “@” mentioned in
  - Beware of @all!

Integration

- Act on behalf of a Webex Teams user
  - Access equivalent to a real spark User (limited by authorized scopes)
- Invoke Webex Teams 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.webex.com/docs/integrations

An integration acts as YOU

and can see and do

the things you can do.

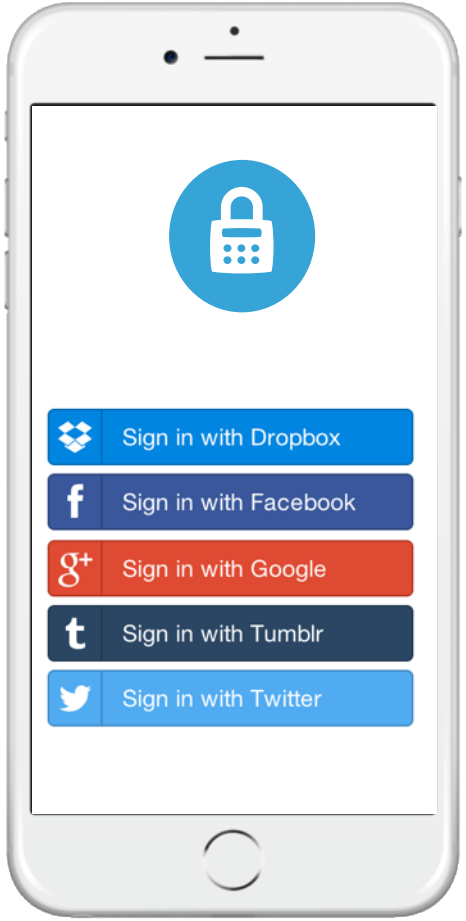
Integrations: Secure with oAuth

Have your app invoke Webex Teams APIs on behalf of the end-user

A personal access token will make calls on your behalf, but in production, you will need your app to post on behalf of others.

To do this in a secure way, Webex Teams supports OAuth2. To achieve this:

- Register an app with Webex
- Request permission using OAuth grant flow
- Exchange the resulting authorization code for an access token
- Use this access token to make your API calls



Access Level of integration

Scopes*

Scopes define the level of access that your integration requires.

Learn more

- [ ] **spark:all**
  Full access to your Webex Teams account
- [ ] **spark:memberships_read**
  List people in the rooms you are in
- [ ] **spark:memberships_write**
  Invite people to rooms on your behalf
- [x] **spark:messages_read**
  Read the content of rooms that you are in
- [x] **spark:messages_write**
  Post and delete messages on your behalf
- [x] **spark:people_read**
  Read your users' company directory
- [x] **spark:rooms_read**
  List the titles of rooms that you are in
- [ ] **spark:rooms_write**
  Manage rooms on your behalf
- [ ] **spark:team_memberships_read**
  List the people in the teams your user belongs to
- [ ] **spark:team_memberships_write**
  Add people to teams on your users' behalf
- [ ] **spark:teams_read**
  List the teams your user's a member of
- [ ] **spark:teams_write**
  Create teams on your users' behalf
- [ ] **spark-admin:licenses_read**
  Access to read licenses available in your user's organizations
- [ ] **spark-admin:metrics_read**
  Access to read metrics in your user's organization
- [ ] **spark-admin:organizations_read**

User permit of Access Level

Cisco Webex

Enter your email address

Email address

Next

Integration Demo
is requesting the following:

- Full access to your Cisco Spark account
- Allow decryption and encryption

Accept

Only ask when requesting new permissions.

Decline

oAuth Authorization Code Flow Summary

1. Application Requests *auth code*

Browser redirect to Webex Authentication

2. Webex returns the *auth code* to application

Browser redirect to Application

3. Request an *access token*

HTTP GET request to Webex API

4. Application gets *access token* and *refresh token*

HTTP GET response from Webex API

Demo: Integration Authorization

- View the static notebooks at https://github.com/jeokrohn/brkcol2175clemea2020/tree/master/Notebook

jupyter Notebook Last Checkpoint: a few seconds ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help Trusted Python 3

Code

Edit Attachments

How to obtain an OAuth refresh token

Create an integration

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

Copy the values for your integration to below code.

The client_secret will only be shown once!

In [1]:

Edit Attachments

```
client_id = 'C7c2beddad408126189a9fb2aefacbb2ae26f2a271b02547419e0b26f48cccbda'
client_secret = 'c804aa273c2efd2e85b192dda2702c5d90ee5fcf0a0c7dcf561ba576a6bceae0'
redirect_uri = 'https://foo.example.com'

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


Deploying an Integration

- Register Integration at https://developer.webex.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: Webex Teams Examples

- View the static notebooks at https://github.com/jeokrohn/brkcol2175clemea2020/tree/master/Notebook

jupyter 2-Webex Teams Last Checkpoint: a day ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

Edit Attachments

Using the Webex Teams APIs

Getting a list of spaces

The endpoint to get a list of spaces is documented here: https://developer.webex.com/endpoint-rooms-get.html. Below is the code to get a list of all spaces.

In [ ]:

Edit Attachments

```python
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()
```


Building a Basic Bot

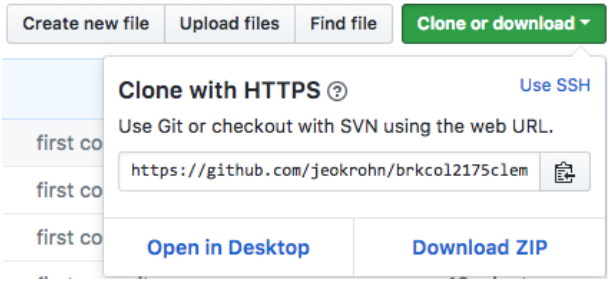
Building a Basic Bot using Python

- Need to start an Ngrok process for redirection of a public URI to our local host
- Use a bot framework to handle POSTs to webhook redirected to local host and to parse the input
- Create handlers for bot commands
- Running Code in the Jupiter Notebook of this session!
- Demo Video available in GitHub repository
- For an alternative way to implement a Bot not requiring a Webhook take a look at: https://github.com/jeokrohn/duwebhook,

Additional Material

Git Repository

- Repository available at: https://github.com/jeokrohn/brkcol2175clemea2020
- Contents:
  - Jupyter notebooks (static)
  - Files required to build Docker image



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/brkcol2175clemea2020
- The GitHub repository allows you to build a Docker image to run a live notebook
- Documentation at GitHub

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

- All videos are published on Box: http://bit.ly/CLEMEABRKCOL2175
- ..together with the PDF of the session presentation

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

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 (easiest option, single command)
  `docker run -it --rm --name jupyter -p 8888:8888 jeokrohn/brkcol2175clemea2020`
- 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)
- Take a look at file `start.sh` on GitHub for information on how to set environment variables for a Docker container

Starting a Docker Container

- Prepared Scripts in GitHub repository:
  - start.sh

```
#!/usr/bin/env bash
docker run -it --rm --name jupyter -p 8890:8888 \
    -e SPARK_CLIENT_SECRET=$SPARK_CLIENT_SECRET \
    -e BOT_ACCESS_TOKEN=$BOT_ACCESS_TOKEN \
    -e BOT_EMAIL=$BOT_EMAIL \
    -e BOT_APP_NAME=$BOT_APP_NAME \
    jeokrohn/brkcol2175clemea2020
```

  - start_mount_local.sh:

```
#!/usr/bin/env bash
docker run -it --rm --name jupyter -p 8890:8888 \
    -e SPARK_CLIENT_SECRET=$SPARK_CLIENT_SECRET \
    -e BOT_ACCESS_TOKEN=$BOT_ACCESS_TOKEN \
    -e BOT_EMAIL=$BOT_EMAIL \
    -e BOT_APP_NAME=$BOT_APP_NAME \
    --mount type=bind,src="$(pwd)/Notebook",dst=/home/jovyan/work/Notebook \
    jeokrohn/brkcol2175clemea2020
```

Conclusion

Follow-Up

- LABCOL-2293: Bots for WebEx Teams - create your own
- 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 Webex Teams 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!
- Webex Teams APIs allow to extend the Webex Teams eco-system
- Start coding!

Complete your online session survey

- Please complete your session survey after each session. Your feedback is very important.
- Complete a minimum of 4 session surveys and the Overall Conference survey (starting on Thursday) to receive your Cisco Live t-shirt.
- All surveys can be taken in the Cisco Events Mobile App or by logging in to the Content Catalog on ciscolive.com/emea.

Cisco Live sessions will be available for viewing on demand after the event at ciscolive.com.

Continue your education

LABCOL-2293: Bots for WebEx Teams - create your own

Demos in the Cisco Showcase

Walk-In Labs

Meet the Engineer 1:1 meetings

Related sessions

Thank you

CISCO

You make **possible**