Agile Management of Webex Calling

There is an API for that

Johannes Krohn

Principal Technical Marketing Engineer

webex by CISCO

webexone 25

Join the conversation!

Scan the QR code to be added to the Webex space for Q&A and more

Agenda

01 Why APIs?

02 Coverage/Capabilities

03 Getting started

04 Use cases/Examples

05 Closing

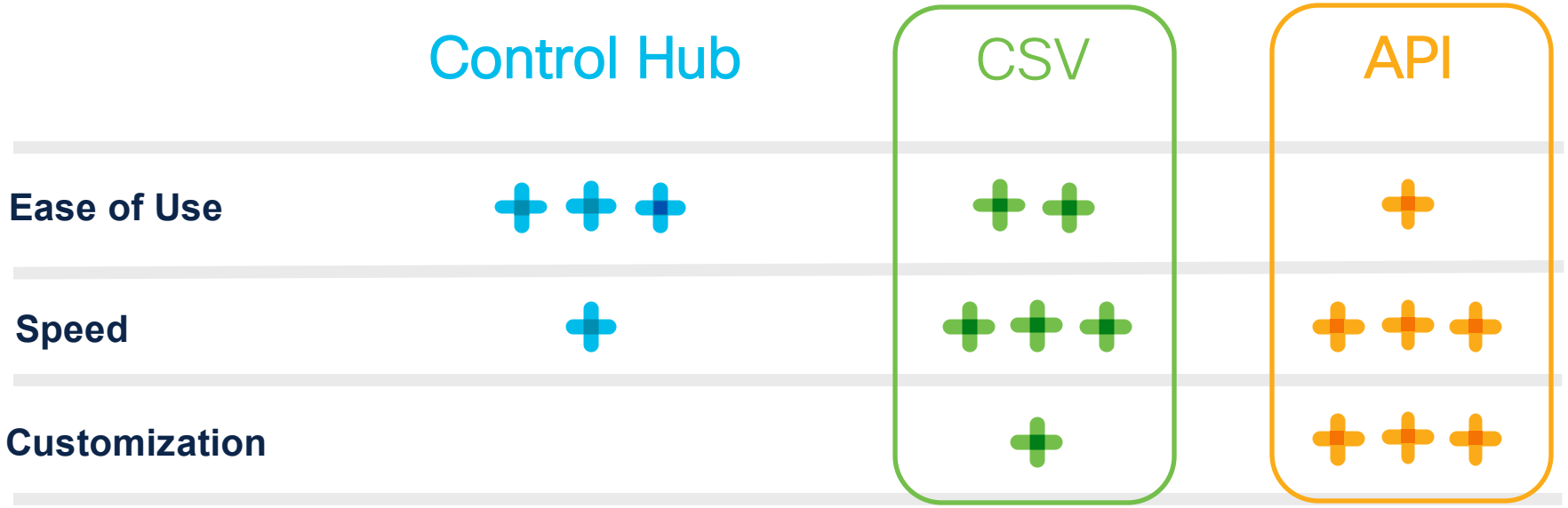
Why APIs?

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

Webex Calling Provisioning Methods



Coverage / Capabilities

Webex APIs

- Documentation: http://developer.webex.com
- Various APIs available:
  - Admin (licenses, locations, memberships, people, ..)
  - Calling (call control, locations, people, org/location settings, ...)
  - Devices (configuration, places, workspace locations, xAPI, ...)
  - Meetings (invitees, participants, preferences, ...)
  - ...
- Comprehensive coverage of Webex Calling
- OAuth access token used for authorization

Webex APIs

+ Admin
+ Calling
+ Contact Center
+ Devices
+ Meetings
+ Messaging
+ Webex Assistant Skills
+ FedRAMP
+ Full API Reference

Webex Calling API Capabilities

- Provisioning
  - Users (incl. calling entitlements), locations (r/o), call pickups, call queues, hunt groups, auto attendant, call parks, schedules, voice messaging settings, …
  - person settings: barge, call forwarding, call intercept, call recording, caller ID, voicemail settings, …
  - Coverage continuously growing*

*Check https://help.webex.com/en-us/article/rdmb0/What's-new-in-Webex-Calling and https://developer.webex.com/docs/api/changelog for updates

- Call Control
  - Dial, answer, reject, hangup, hold/resume, mute/unmute, divert, transfer, park/retrieve, start/stop/pause/resume recording, DTMF, push, pickup, barge
- Webhook Notifications/Events
  - Voice messages
  - Call events

References:
https://developer.webex.com/calling/docs/webex-calling-overview
https://developer.webex.com/blog/calling-apis-overview

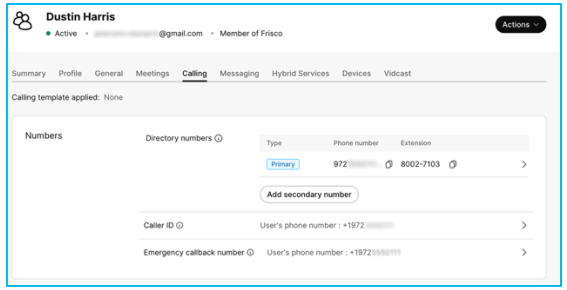
Webex Calling Provisioning APIs

- Locations, https://developer.webex.com/calling/docs/api/v1/locations
  - List locations
  - Get location details
  - Create/Update locations
  - Delete: not possible
- People, https://developer.webex.com/admin/docs/api/v1/people*
  - List
  - CRUD
  - `callingData` parameter to access calling data (set, modify TN/extension)

*For user management use of the SCIMv2 user API is preferred:
https://developer.webex.com/admin/docs/api/v1/scim-2-users
Preferred endpoint for TN/extension assignment/updates:
PATCH /licenses/users

Webex Calling Provisioning APIs

- Person Settings,
  https://developer.webex.com/calling/docs/api/v1/user-call-settings-1-2
  https://developer.webex.com/calling/docs/api/v1/user-call-settings-2-2
  - Settings found in person's Calling tab in Control Hub



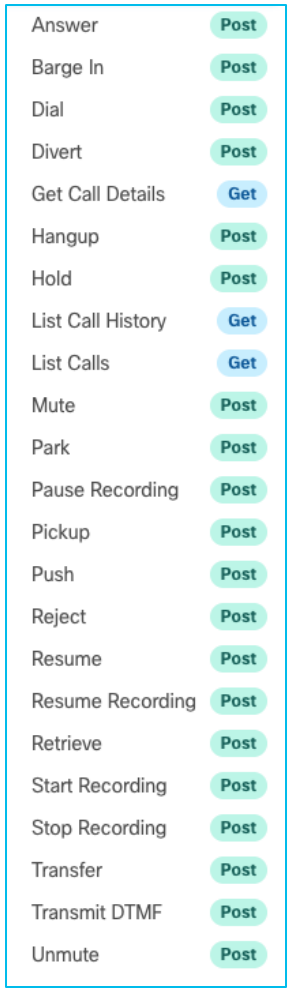
- Comprehensive coverage

API REFERENCE

- All APIs
  - Call Controls
  - Call Routing
  - Call Settings For Me
  - Calling Service Settings
  - Client Call Settings
  - Conference Controls
  - Contact Center with Calling For Me
  - Converged Recordings
  - DECT Devices Settings
  - Device Call Settings
  - Devices
  - Emergency Services Settings
  - Features: Auto Attendant
  - Features: Call Park
  - Features: Call Pickup
  - Features: Call Queue
  - Features: Hunt Group
  - Features: Paging Group
  - Features: Announcement Playlist
  - Features: Announcement Repository
  - Features: Call Recording
  - Features: Customer Experience Essentials
  - Features: Hot Desking Sign-in via Voice Portal
  - Features: Operating Modes
  - Features: Single Number Reach
  - Features: Virtual Extensions
  - Location Call Settings
  - Location Call Settings: Schedules
  - Location Call Settings: Voicemail
  - Location Call Settings: Call Handling
  - Locations
  - Numbers
  - Partner Reports/Templates
  - People
  - PSTN
  - Recording Report
  - Reports
  - Reports: Detailed Call History
  - Send Activation Email
  - User Call Settings (1/2)
  - User Call Settings (2/2)
  - Virtual Line Call Settings
  - Workspace Call Settings (1/2)
  - Workspace Call Settings (2/2)
  - Workspaces

Webex Calling Call Controls

- Actions
  - Dial, answer, reject, hangup, hold/resume, divert, transfer, park/retrieve, start/stop/pause/resume recording, DTMF, push, pickup, barge
- Management
  - List, get details, call history
  - List/Details use common call object
- Requires user access token
  - No org level (admin) operations

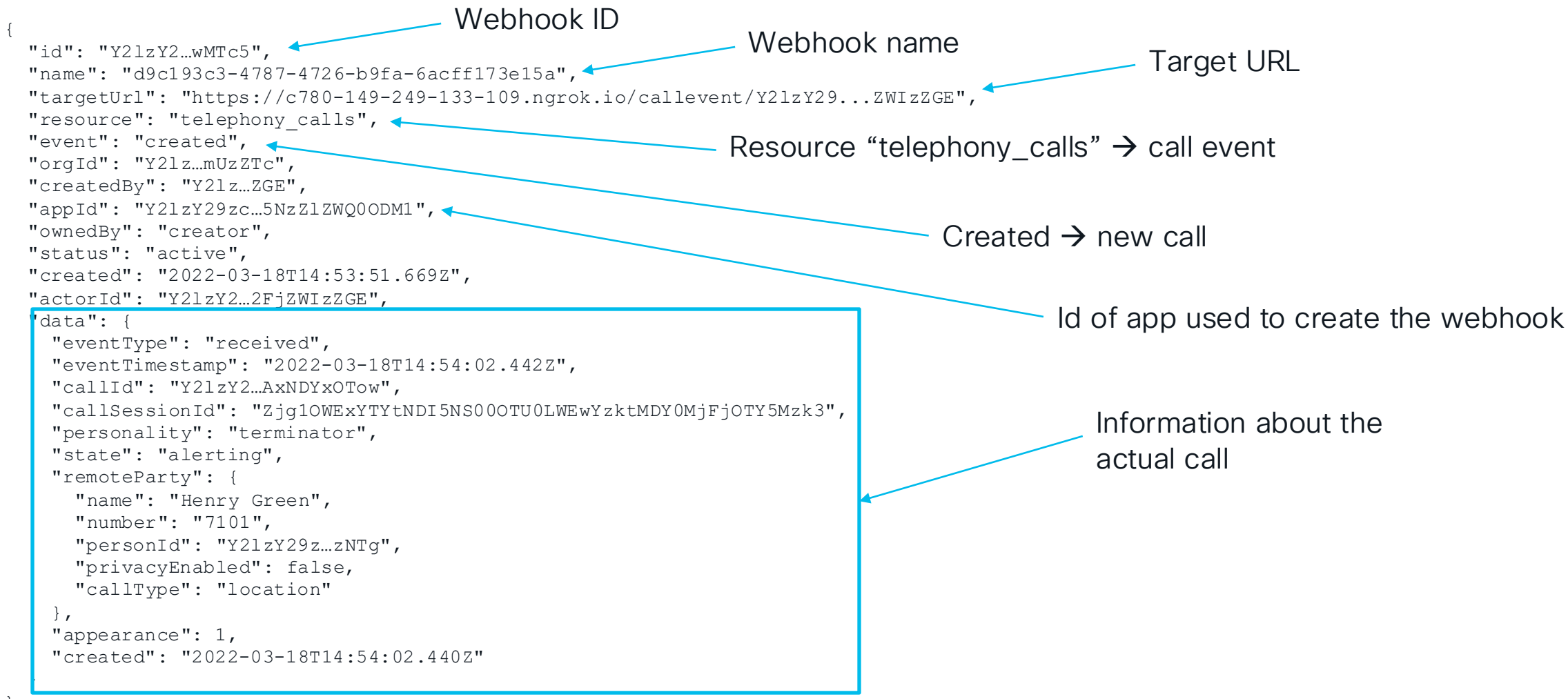


Webhook Notifications/Events

- Webhook API to manage webhooks: https://developer.webex.com/meeting/docs/api/v1/webhooks
- Resource: `telephony_calls`
- Events: `created`, `updated`, `deleted`

https://developer.webex.com/docs/api/guides/webhooks

Telephony_call event example



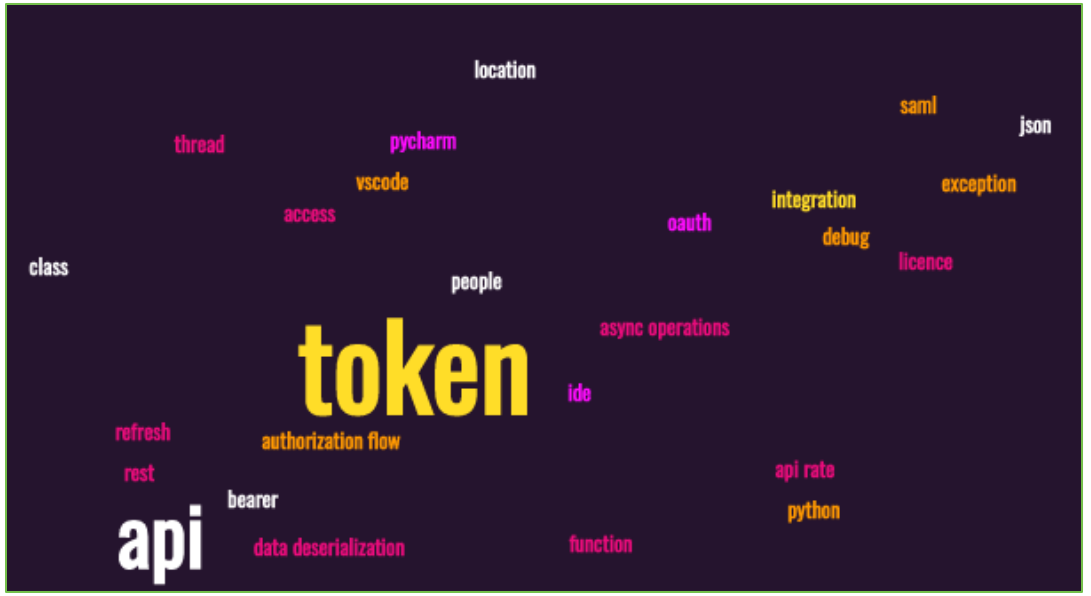
Webex Calling APIs Overview

	PROVISIONING	CALL CONTROL	ANALYTICS & REPORTING
Customer Journey	Setup, Onboard, Manage	Call, Meet, Collaborate	Achieve Customer Success
Representative Tasks	• Manage users, phone #s, locations, & services • Assign licenses • Create and manage location features	• Place, answer, hang up calls • Stop / start / pause recording • Transmit DTMF digits • List active calls / get history	• Detailed call records • Onboarding, usage, & quality reporting • Automated reporting setup
Sample Solutions	• Installation, activation, & onboarding • Ongoing services management & care • Self-service via partner portal	• Custom enterprise calling integrations • Cloud business platform integration • Custom app development	• User training & adoption services • Business process design & optimization • Vertical solutions design & oversight

Getting Started

Using Webex APIs

- Documentation at: https://developer.webex.com/
- But: Steep learning curve
- A lot of concepts to master
- SDK helps to abstract from the “dirty details”



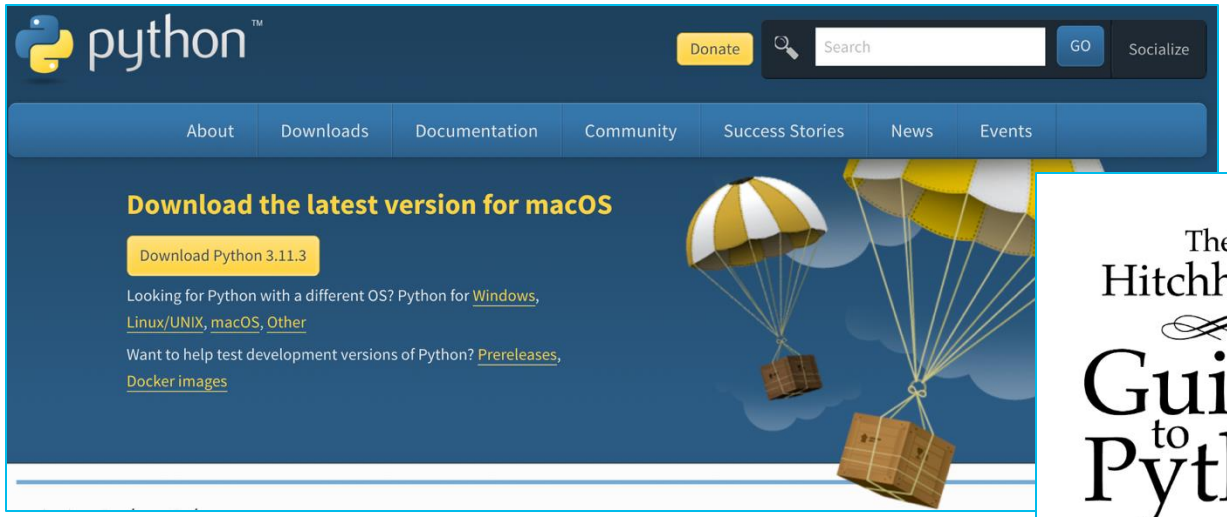
Developer Sandbox

- Sandbox
  - playground to test API calls
  - Avoid impact on production org
- Limited to 10 users
- Allows to test capabilities not available w/ Webex free plans
- No Cisco PSTN
  - Can add on-premises PSTN (Local Gateway) for PSTN access

https://developer.webex.com/admin/docs/developer-sandbox-guide

Installing Python

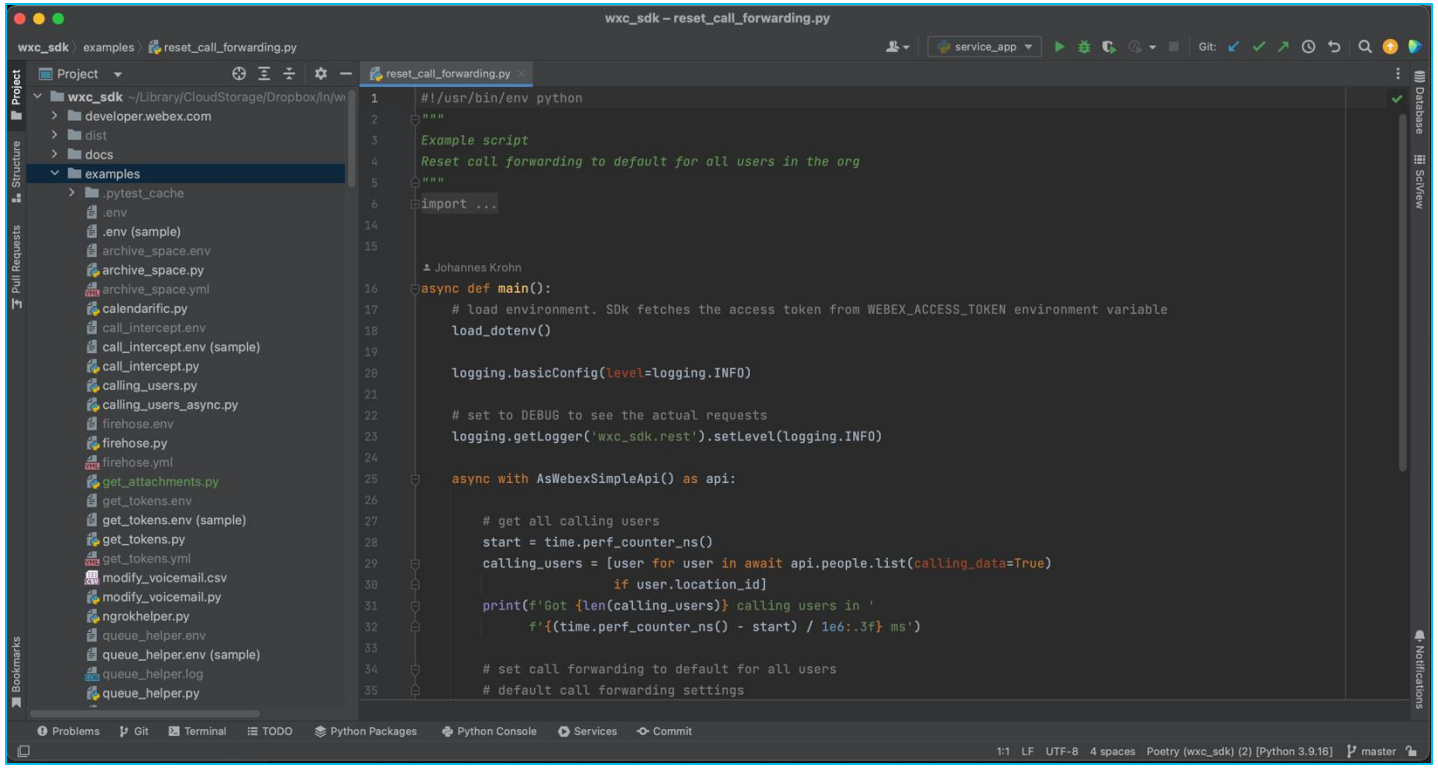
- Installers are are available at https://www.python.org/downloads/
- Mac tip: install Python via Homebrew: https://docs.python-guide.org/starting/install3/osx/
  - Avoids issues with GNU readline (for example when using https://pypi.org/project/cmd2/)



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

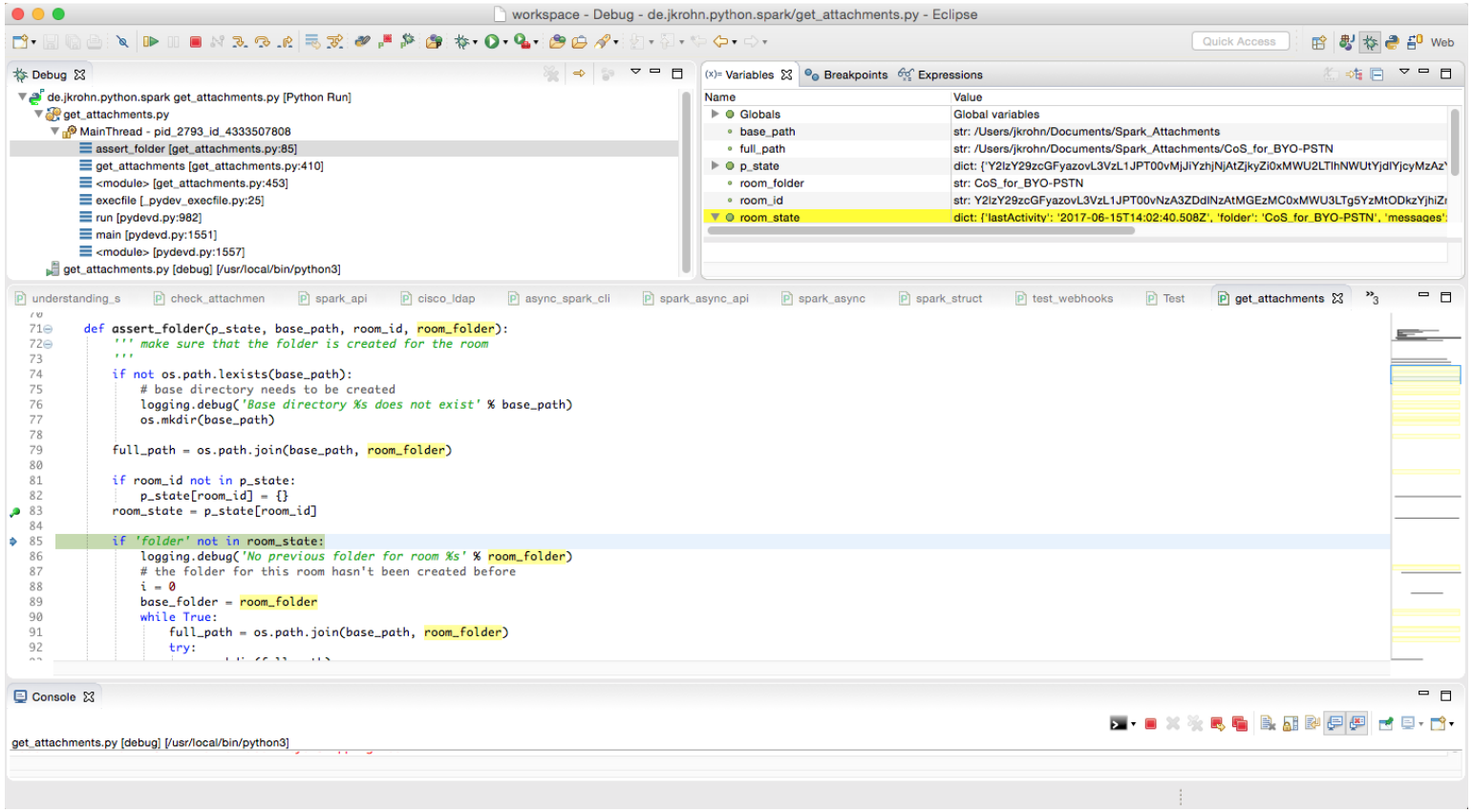
- What Do you prefer?
- Or this?

```
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

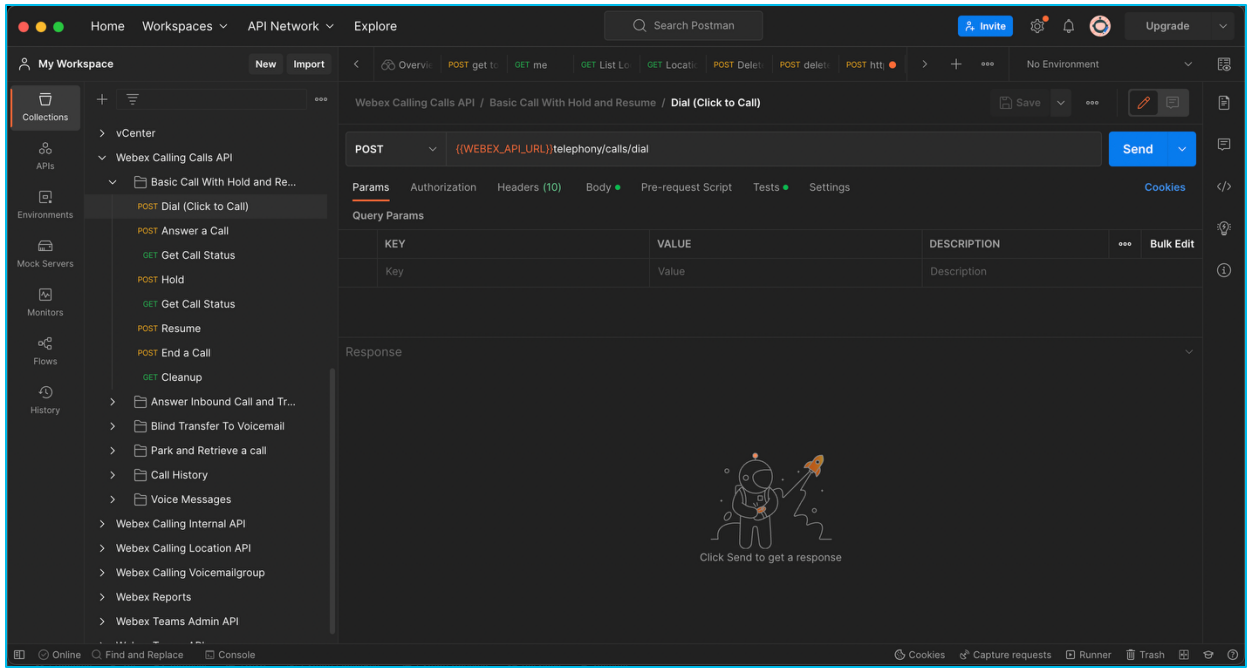
- IDLE (Standard IDE)
- PyCharm
- VS Code
- PythonAnywhere

PyCharm

pythonanywhere

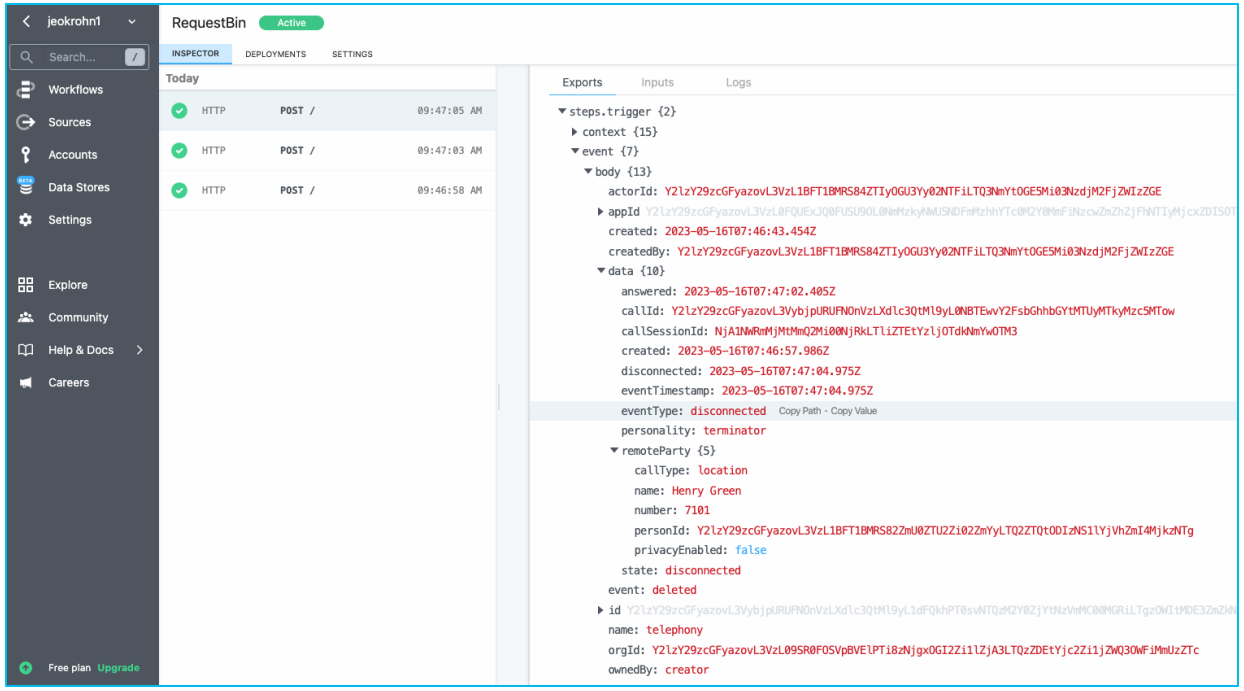
Postman: Test APIs

- 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 Messaging and Admin APIs: https://github.com/CiscoDevNet/postman-webex



RequestBin: See Webhooks in Action

- Free service: https://pipedream.com/requestbin
- Creates unique URL
- Use case: Webex webhook pointing to Requestbin to test webhook operation
- Provides real-time view on requests hitting the URL



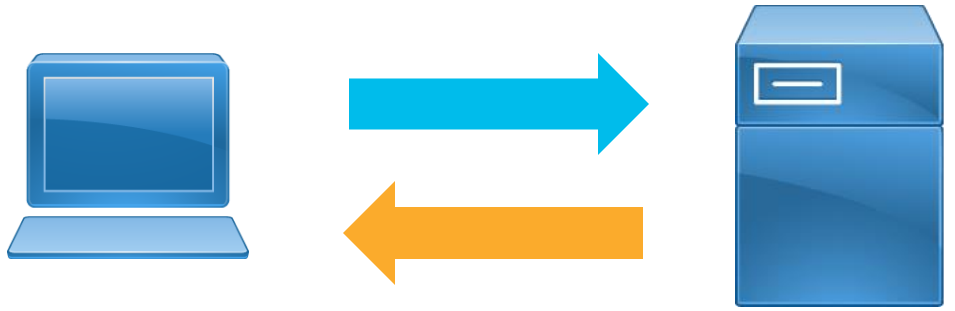
GitHub

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

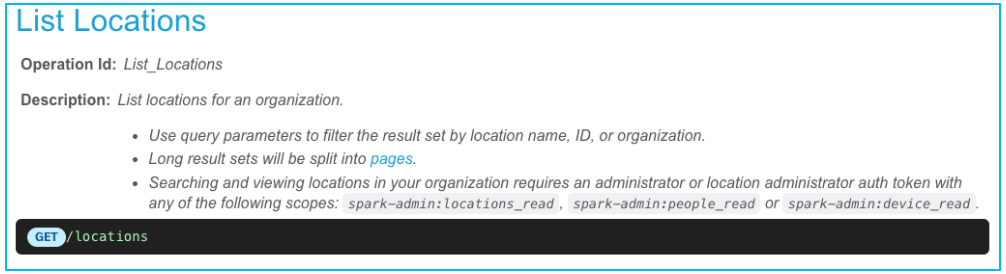
Consuming APIs

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



Calling a Webex API Endpoint
Listing Webex Calling Locations



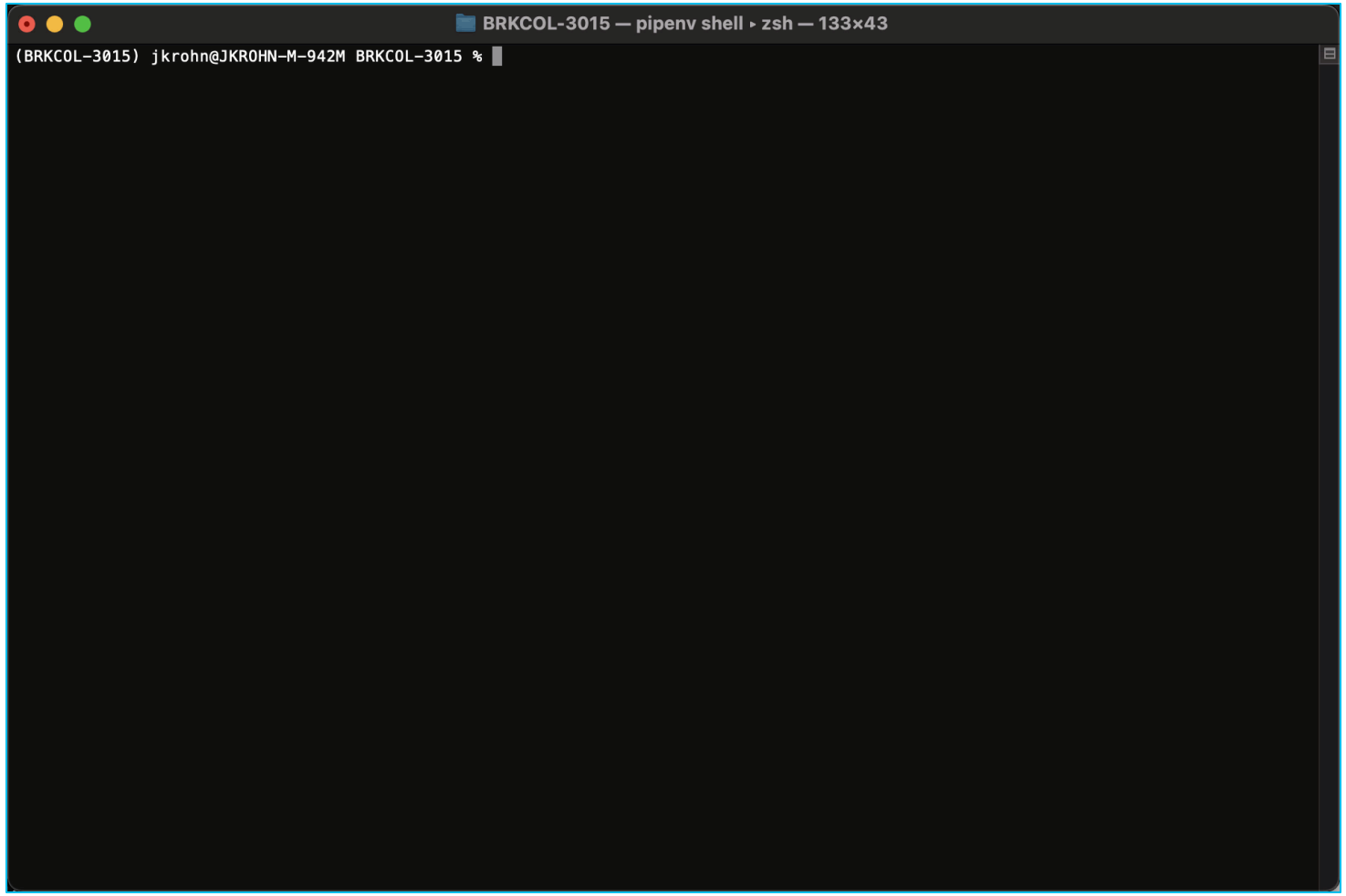
https://developer.webex.com/calling/docs/api/v1/locations/list-locations

```python
def main():
    # load .env file
    load_dotenv()

    # after reading .env file all variables defined in the file are accessible as environment variables
    access_token = os.getenv('WEBEX_TOKEN')
    if access_token is None:
        raise

    url = 'https://webexapis.com/v1/locations'
    with requests.Session() as session:
        headers = {'Authorization': f'Bearer {access_token}'}
        response = session.get(url=url, headers=headers)
        response.raise_for_status()
        data = response.json()
        print(f'{len(data["items"])} locations found')
        for location in data['items']:
            print(location)

        # look for locations in California
        ca_locations = [location for location in data['items']
                        if location['address']['state'] == 'CA']
        print(f'{len(ca_locations)} locations in CA')
        print(', '.join(loc['name'] for loc in ca_locations))
```

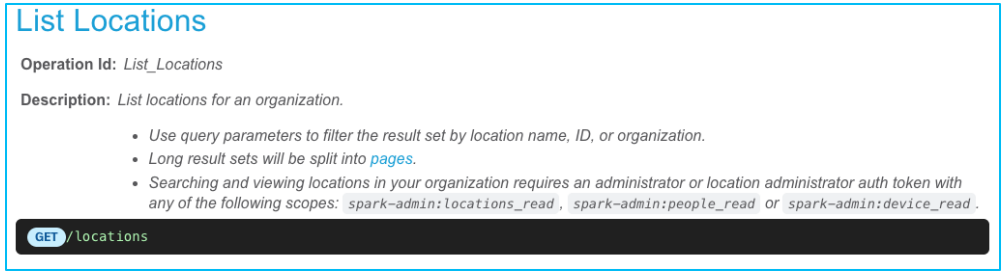
- URL of the endpoint
- Session() from requests module is used
- Fabricate the Authorization header
- Call the endpoint
- Check for errors
- Parse the JSON response into a dict
- Accessing the response values as dict keys

https://github.com/jeokrohn/BRKCOL-3015/blob/main/list_locations_direct.py

Calling a Webex API Endpoint
Listing Webex Calling Locations



Calling a Webex API Endpoint
Listing Webex Calling Locations



https://developer.webex.com/calling/docs/api/v1/locations/list-locations

```python
def main():
    # load .env file
    load_dotenv()

    # after reading .env fi
    access_token = os.geten
    if access_token is None
        raise

    url = 'https://webexapi
    with requests.Session()
        headers = {'Authori
        response = session.
        response.raise_for_
        data = response.jso
        print(f'{len(data["
        for location in dat
            print(location)

        # look for locations in California
        ca_locations = [location for location in data['items']
                        if location['address']['state'] == 'CA']
        print(f'{len(ca_locations)} locations in CA')
        print(', '.join(loc['name'] for loc in ca_locations))
```

That was easy, but..

- Accessing dictionary values by key is hard and error prone
- Missing handling of 429 responses (throttling)
- Missing pagination handling
- Handling of additional parameters (name, id)

There **has** to be a better way?!

…ests module is used

…rization header

…ponse into a dict

…nse values as dict keys

wxc_sdk: SDK for Webex Calling APIs

- PyPi: https://pypi.org/project/wxc-sdk/
- Homepage: https://github.com/jeokrohn/wxc_sdk
- Documentation: https://wxc-sdk.readthedocs.io/en
- Simple SDK to work with Webex APIs
  - Focus on Webex Calling specific endpoints … and more
- Takes care of all the “ugly” stuff
  - JSON (de-)serialization, authentication, 429 retries,
  - Pagination, …
  - Logging
- Python objects for all API objects
  - Tab completion → efficient coding
- Actively maintained
  - New API endpoints will be added continuously
- Foundation for your provisioning automation and other projects around Webex Calling

```python
#!/usr/bin/env python
"""
Demonstration of how to call a Webex API endpoint using the SDK
"""
import os

import wxc_sdk
from dotenv import load_dotenv


def main():
    load_dotenv()

    # after reading .env file all variables defined in the file are accessible as environment variables
    access_token = os.getenv('WEBEX_TOKEN')

    with wxc_sdk.WebexSimpleApi(tokens=access_token) as api:
        locations = list(api.locations.list())
        print(f'{len(locations)} locations found')
        for location in locations:
            print(location)

        ca_locations = [location for location in locations
                        if location.address.state == 'CA']
        print()
        print(f'{len(ca_locations)} locations in CA')
        print(', '.join(loc.name for loc in ca_locations))


if __name__ == '__main__':
    main()
```

Calling a Webex API Endpoint
Listing Webex Calling Locations using the SDK

```python
#!/usr/bin/env python
"""
Demonstration of how to call a Webex API endpoint using the SDK
"""
import os

import wxc_sdk
from dotenv import load_dotenv


def main():
    load_dotenv()

    # after reading .env file all variables defined in the file are accessible as environment variables
    access_token = os.getenv('WEBEX_TOKEN')

    with wxc_sdk.WebexSimpleApi(tokens=access_token) as api:
        locations = list(api.locations.list())
        print(f'{len(locations)} locations found')
        for location in locations:
            print(location)

        ca_locations = [location for location in locations
                        if location.address.state == 'CA']
        print()
        print(f'{len(ca_locations)} locations in CA')
        print(', '.join(loc.name for loc in ca_locations))


if __name__ == '__main__':
    main()
```

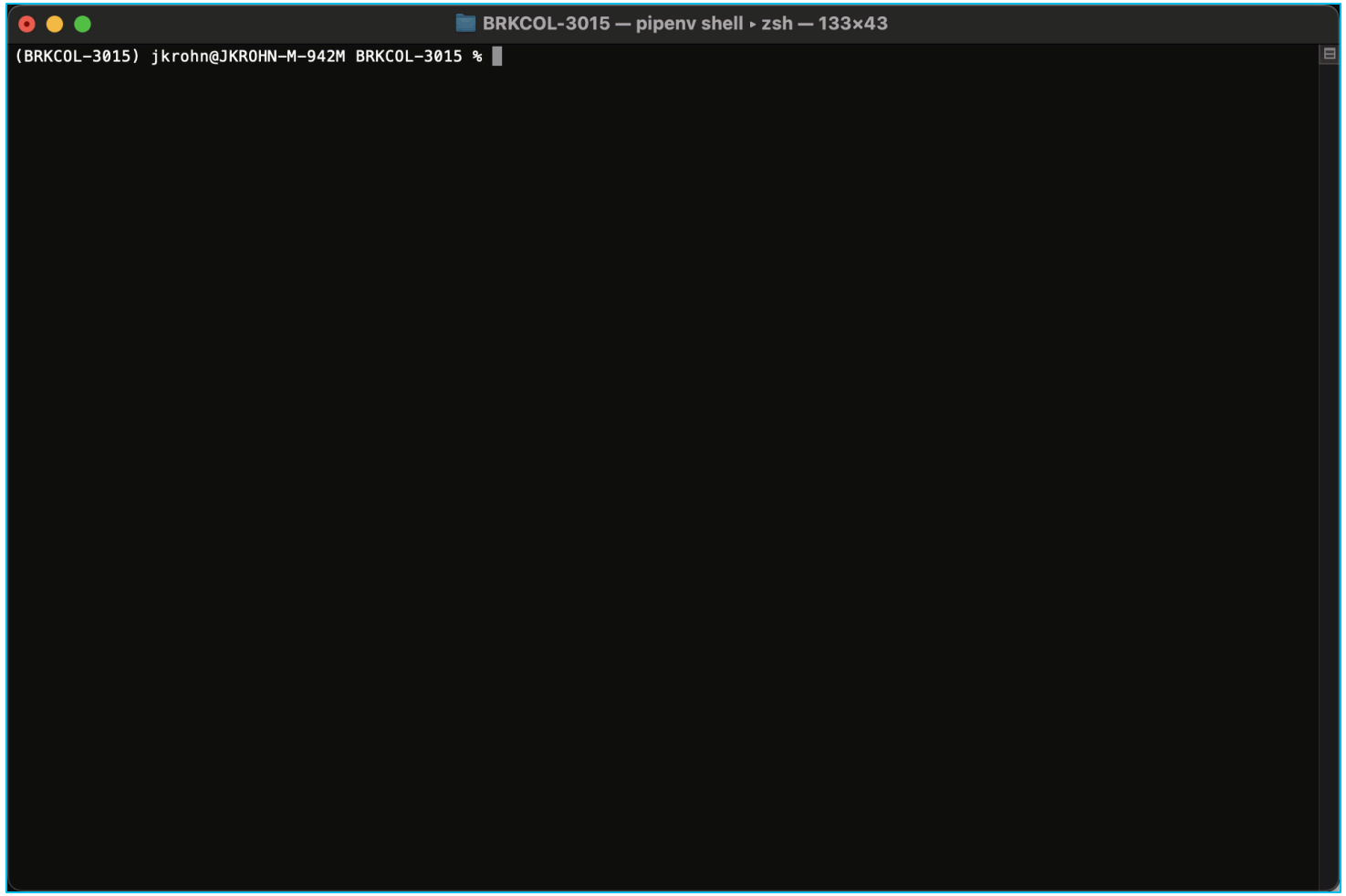
The API object

Get list of locations

Access data using attributes of Python classes

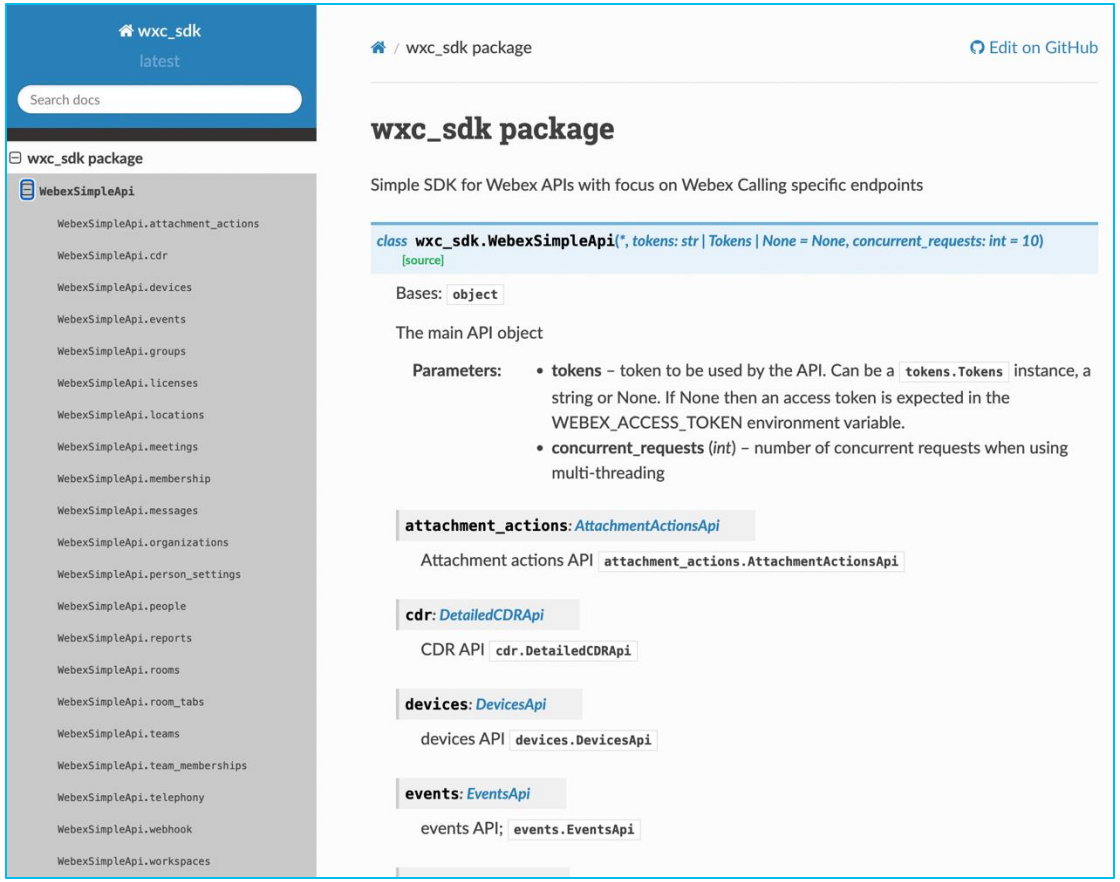
https://github.com/jeokrohn/BRKCOL-3015/blob/main/list_locations_sdk.py

Calling a Webex API Endpoint
Listing Webex Calling Locations using the SDK



wxc_sdk: Comprehensive Coverage

- SDK covers all Webex Calling specific API endpoints
- Additionally:
  - Licenses, memberships, messages, people, teams, team memberships, webhooks, ...
- Available methods:
  - https://wxc-sdk.readthedocs.io/en/1.26.0/user/method_ref.html
- Easy token management



https://wxc-sdk.readthedocs.io/en

Tokens

Tokens
Why and How?

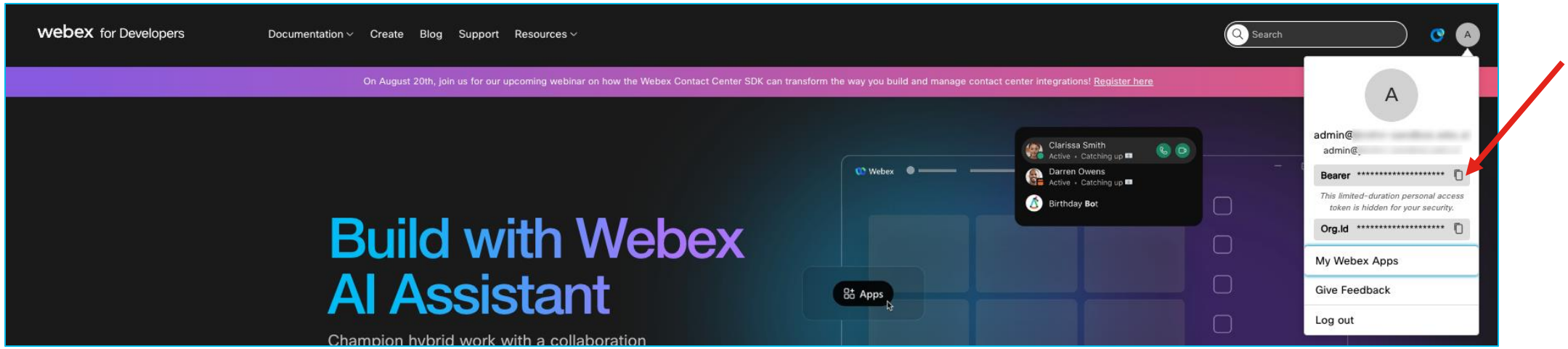
- Access tokens are required to authorize API accces
- .. Can be obtained in different ways:
  - Personal access token (developer token)
  - Integration (OAuth2 authorization flow)
  - Service App
- NEVER(!!!) store tokens in your source files
- NEVER(!!!) push tokens to GitHub repositories
- NEVER(!!!) share tokens in any shape or form
- Best practice:
  - In your code read access token from environment variable
  - Use `dotenv.loadenv()` to load `.env` file with environment variables
  - Exclude `.env` from version control (Git) by adding exclusion in `.gitignore`
  - Integration tokens can be cached in local files .. but make sure to restrict access and not push to GitHub

```
GET https://webexapis.com/v1/people
User-Agent: python-requests/2.30.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
Authorization: Bearer MGY4MzNjNzgt***
content-type: application/json;charset=utf-8
```

https://developer.webex.com/calling/docs/getting-started

Personal Access Token

- From developer.webex.com
- Limited lifetime (12 h)
- Should NEVER be used in production
- Testing only



Integration – The Better Way to Obtain Tokens

- Act on behalf of a Webex user
  - Access equivalent to a real Webex user (limited by authorized scopes)
- Invoke Webex 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

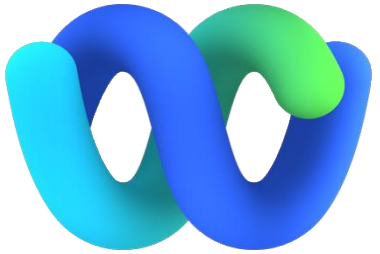
OAuth Authorization Code Flow

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

Integration Tokens in Scripts Using wxc_sdk

- `wxc_sdk` offers an easy way to work with cached tokens in scripts
- Cache tokens in YML file
- Get tokens from OAuth flow redirecting to http://localhost:6001/redirect
- Spin up temporary (primitive) server to handle final step of OAuth flow

Prepare integration based on values read from environment

Read, create, refresh, cache tokens

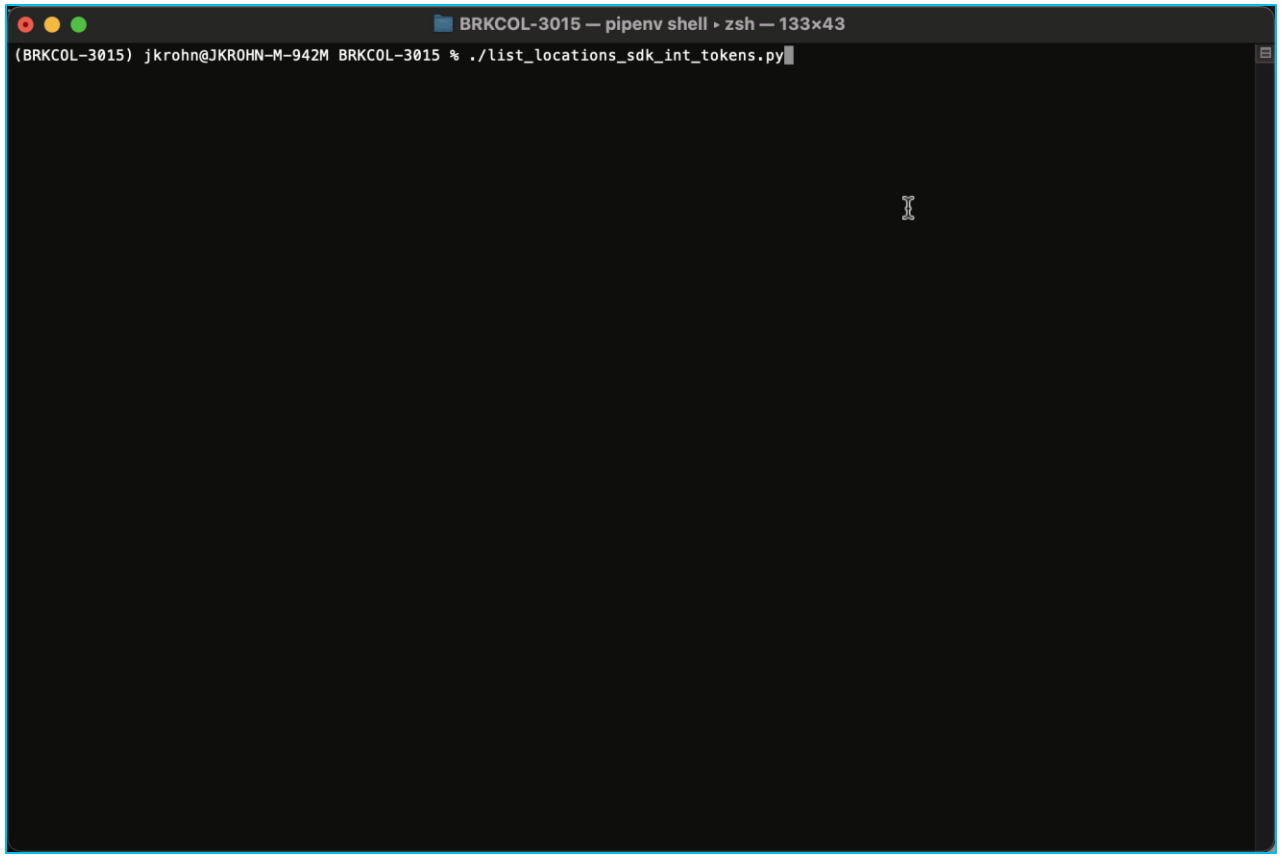
```
access_token: ZGVlY2***
expires_at: '2023-05-30T14:35:37.246110+00:00'
refresh_token: NTlk***
refresh_token_expires_at: '2023-08-14T14:08:57.246110+00:00'
token_type: Bearer
```

```python
#!/usr/bin/env python
"""
Demonstration of how to call a Webex API endpoint using the SDK with cached integration tokens
"""
import os
from os.path import splitext, basename

from dotenv import load_dotenv

from wxc_sdk import WebexSimpleApi
from wxc_sdk.integration import Integration
from wxc_sdk.scopes import parse_scopes


def get_tokens():
    """
    get (cached) integration tokens
    """
    env_vars = ('INTEGRATION_CLIENT_ID', 'INTEGRATION_CLIENT_SECRET', 'INTEGRATION_SCOPES')
    if not all(os.getenv(s) for s in env_vars):
        raise KeyError(f'Not all required environment variables ({", ".join(env_vars)}) defined.')

    client_id = os.getenv('INTEGRATION_CLIENT_ID')
    client_secret = os.getenv('INTEGRATION_CLIENT_SECRET')
    scopes = parse_scopes(os.getenv('INTEGRATION_SCOPES'))
    integration = Integration(client_id=client_id,
                              client_secret=client_secret,
                              scopes=scopes,
                              redirect_url='http://localhost:6001/redirect')
    yml_path = f'{splitext(basename(__file__))[0]}.yml'
    tokens = integration.get_cached_tokens_from_yml(yml_path=yml_path)
    return tokens
```

https://github.com/jeokrohn/BRKCOL-3015/blob/main/list_locations_sdk_int_tokens.py

Integration Tokens in Scripts Using wxc_sdk

- Without cached tokens OAuth flow gets initiated
- Auth code exchanged for tokens
- Tokens are cached
- Next execution uses cached tokens



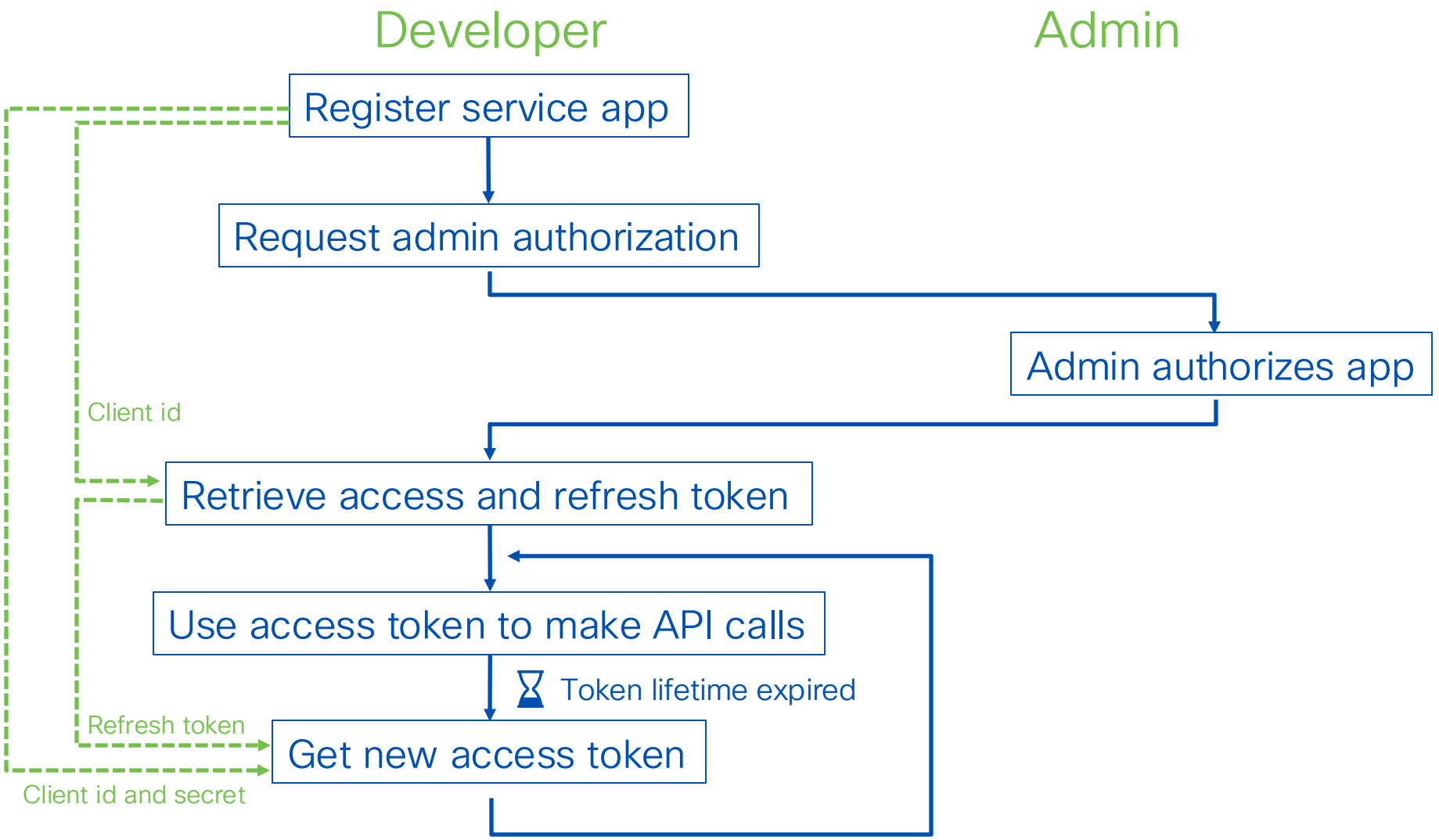
https://github.com/jeokrohn/BRKCOL-3015/blob/main/list_locations_sdk_int_tokens.py

Service Apps

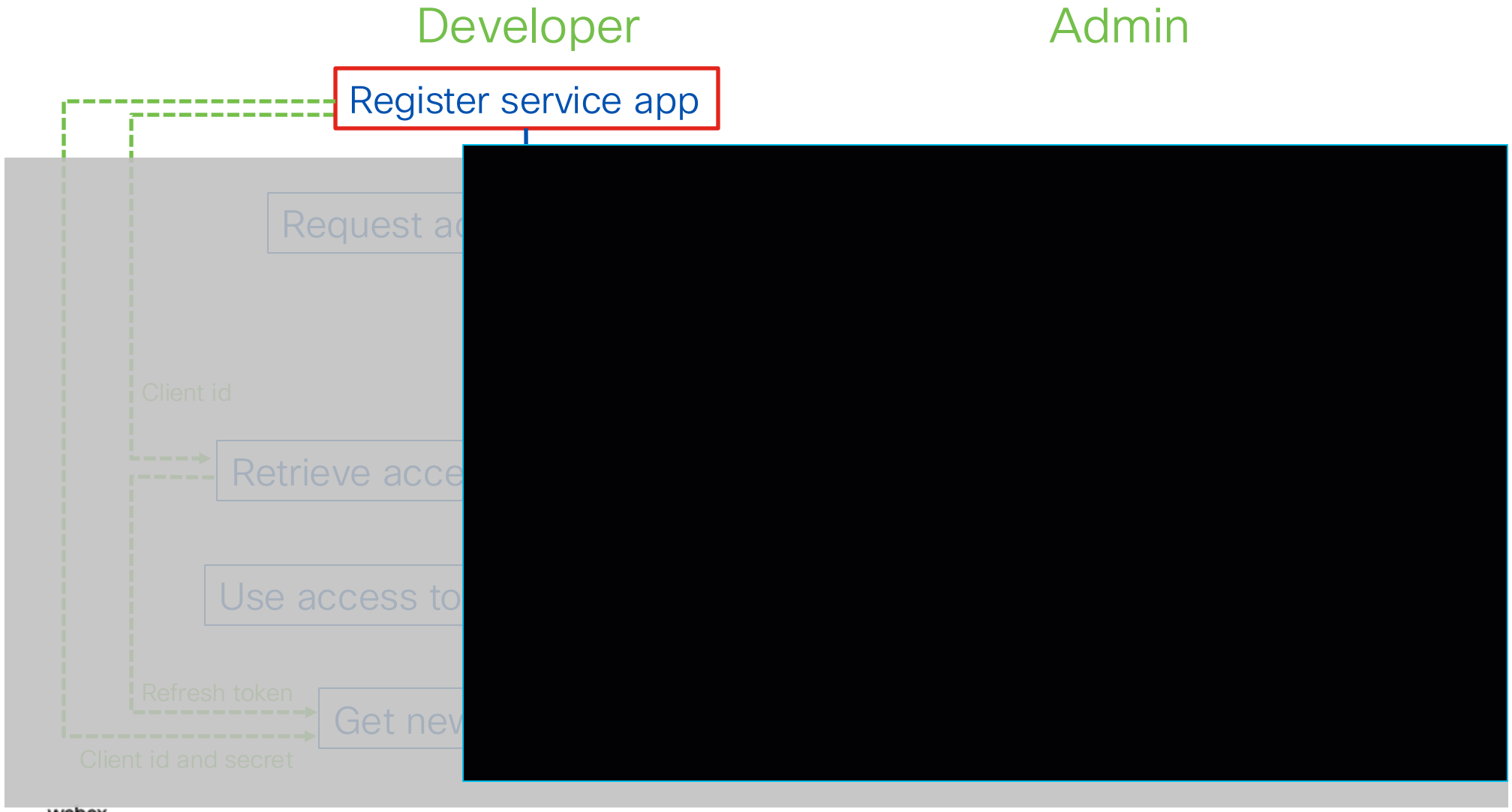
- Machine account
- Request admin permission independent of user account
- Use cases
  - Provisioning
  - Reporting
  - Scheduling systems
  - ...
- Similar to integrations .. but no user specific authorization flow

https://developer.webex.com/admin/docs/service-apps

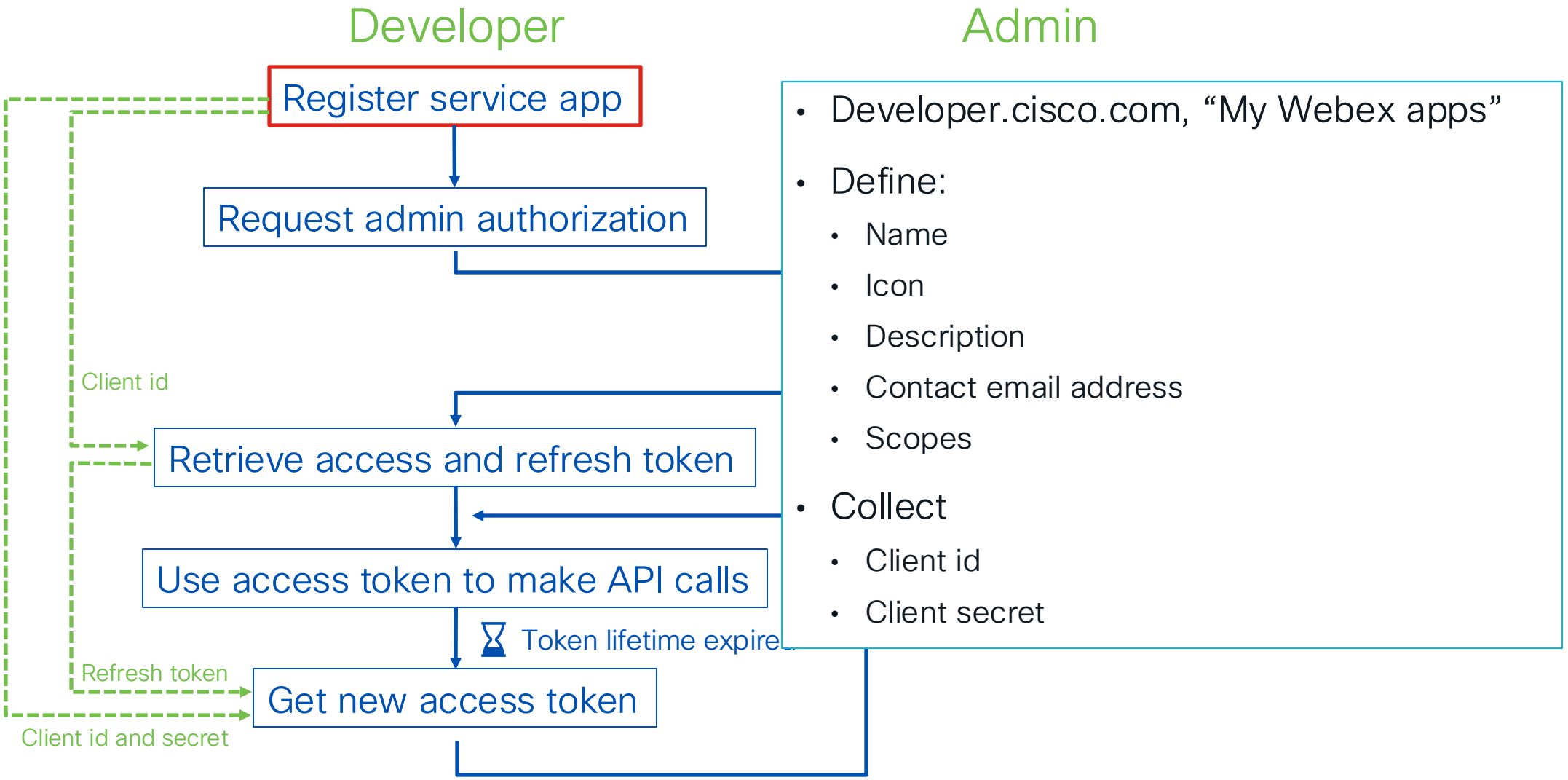
Using Service App Tokens



Using Service App Tokens



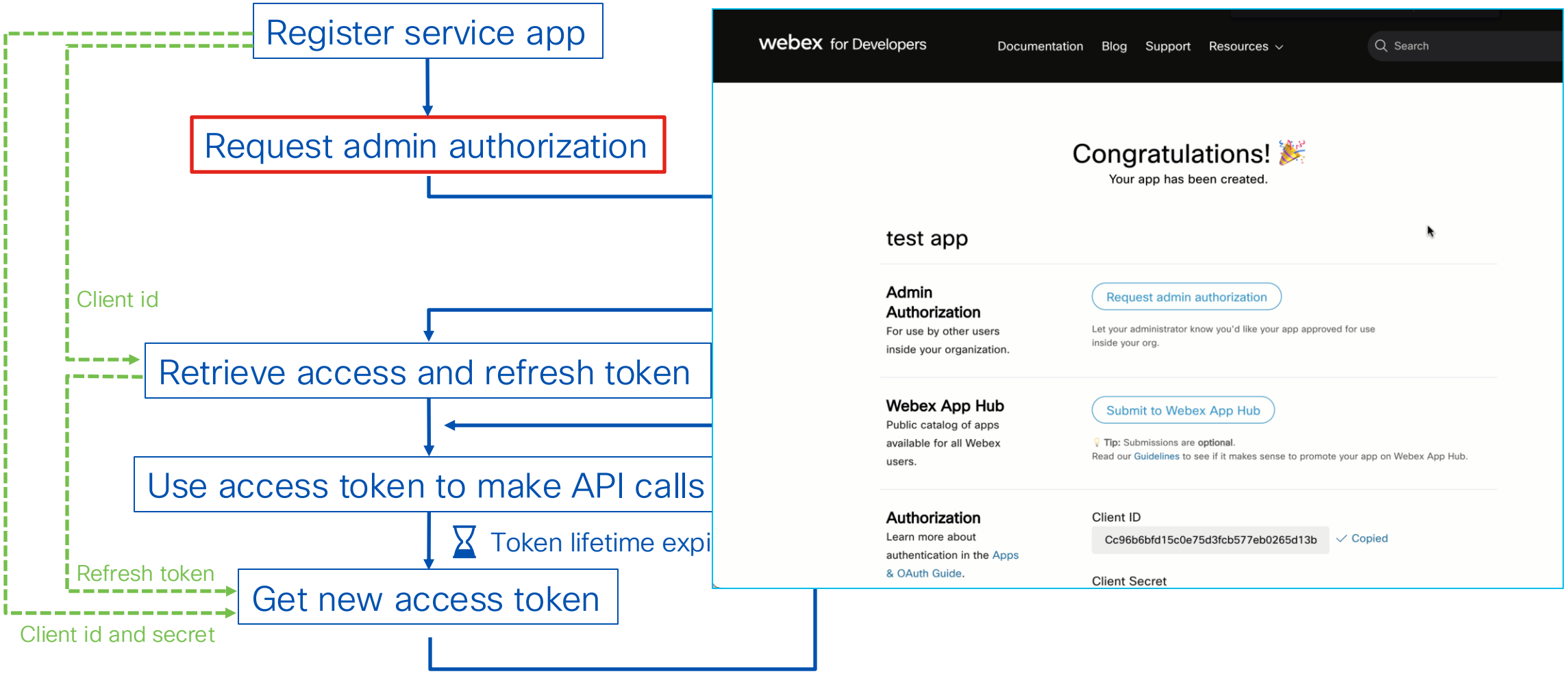
Using Service App Tokens



Using Service App Tokens

Developer

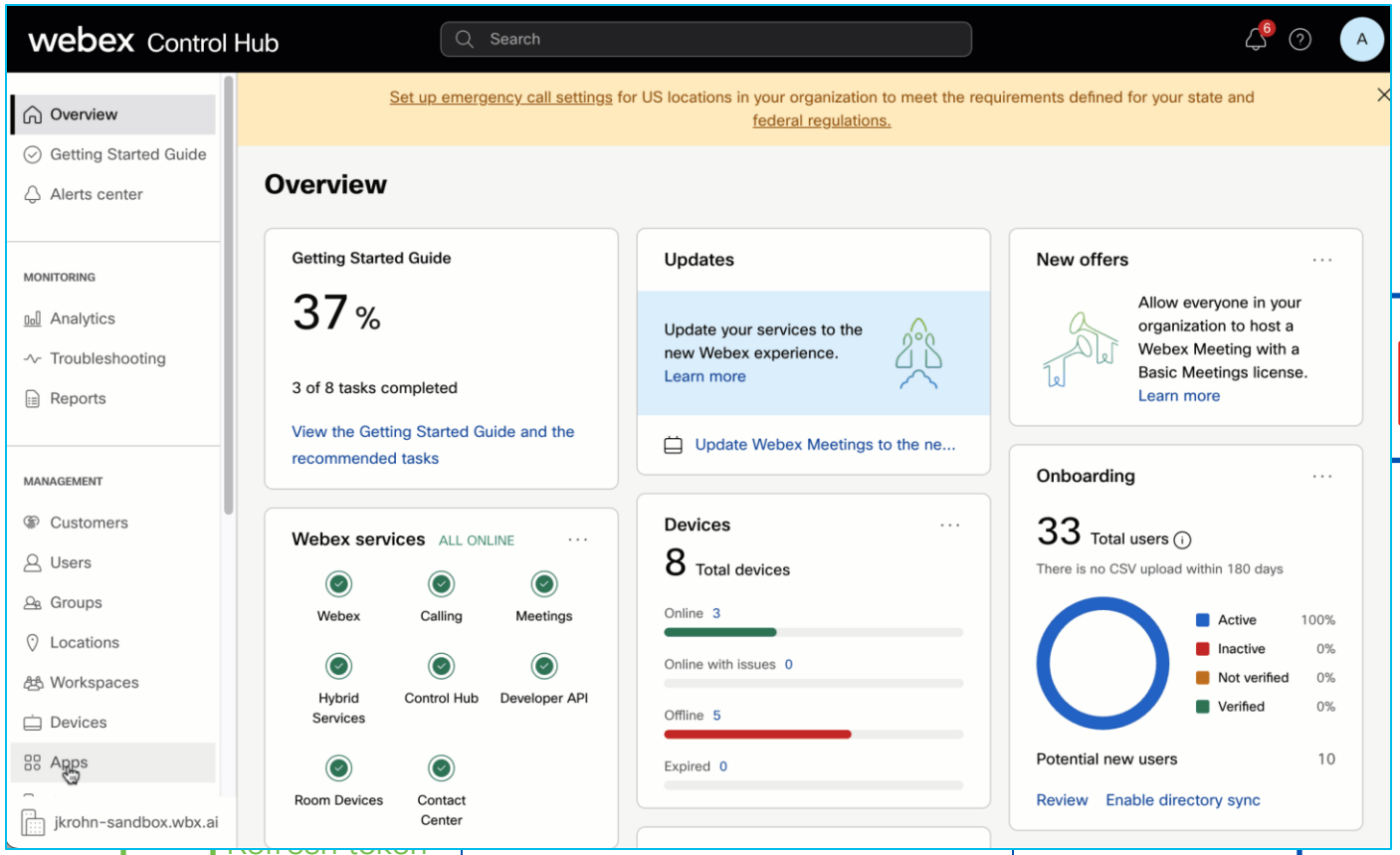
Admin



Using Service App Tokens

Developer

Admin



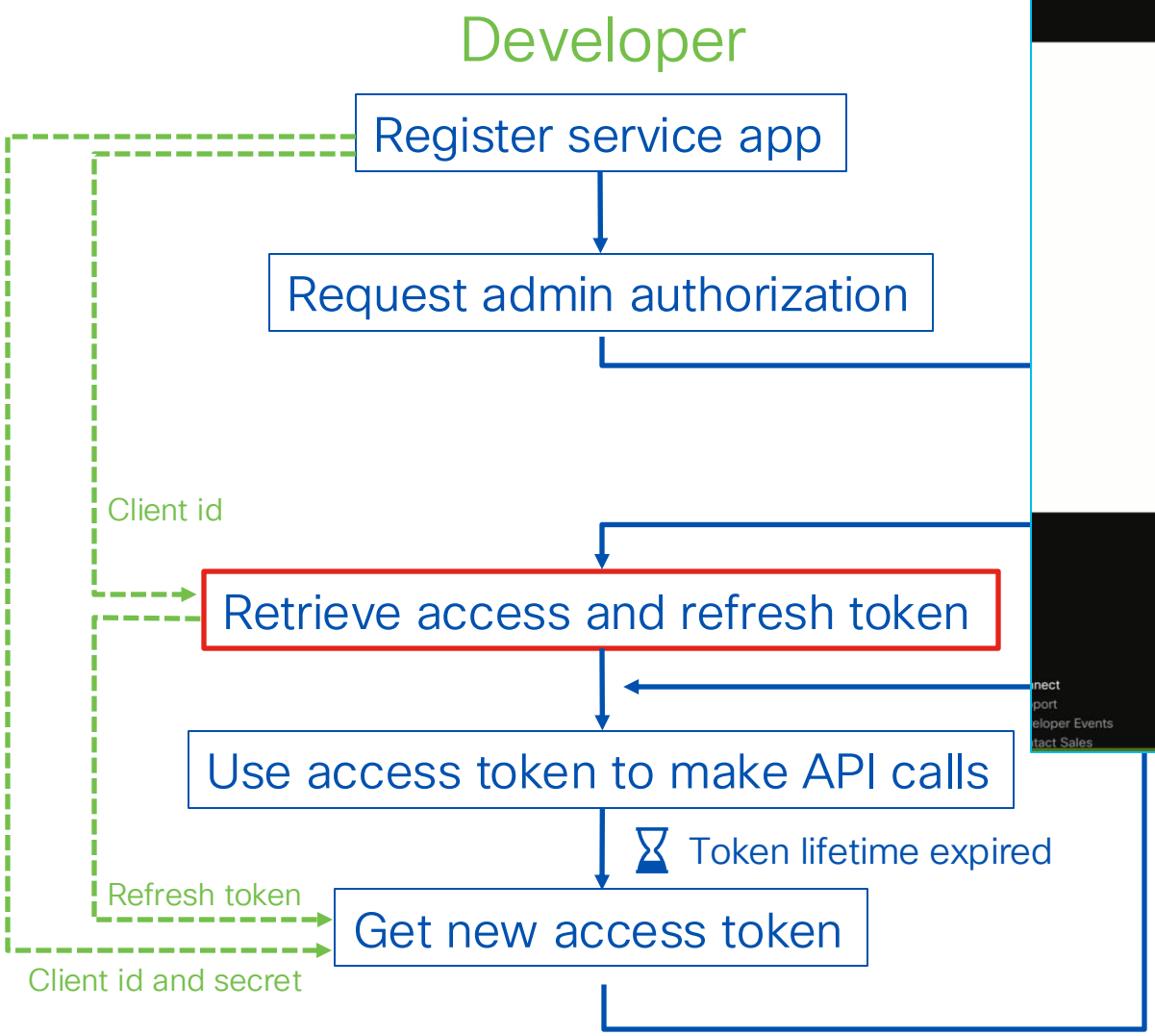
Admin authorizes app

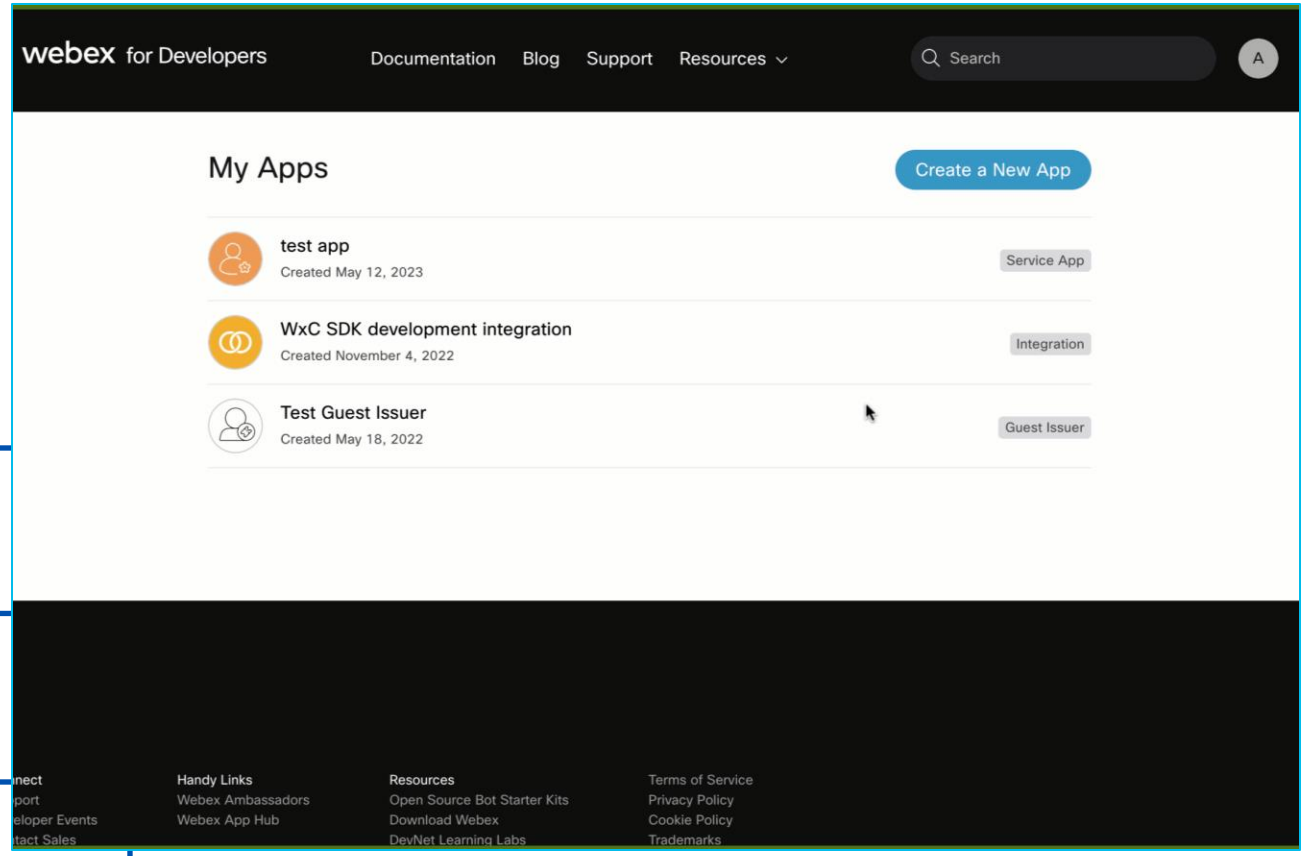
- "Apps" section in Control Hub
- Review scopes
- Authorizing user is documented

Get new access token

Client id and secret

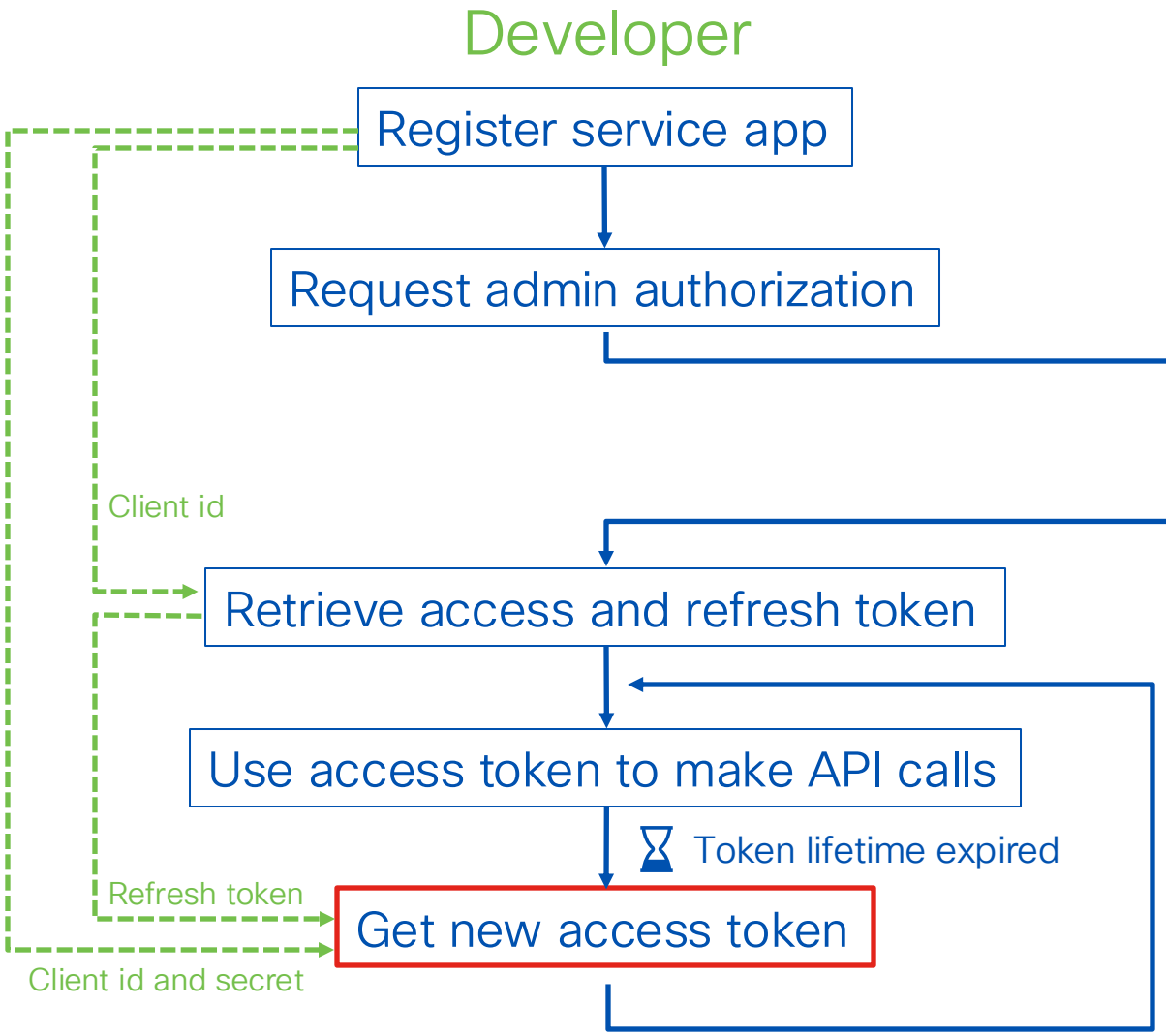
Using Service App Tokens





- Developer requests tokens for authorized org
- Client secret needed to authorize the request

Using Service App Tokens



- When getting close to token lifetime
- New access token can be obtained using the refresh token

```
POST https://webexapis.com/v1/access_token
Content-Type: application/x-www-form-urlencoded
  --- body ---
  grant_type: refresh_token
  client_id: ***
  client_secret: ***
  refresh_token: ***
Response
Content-Type: application/json
--- response body ---
  {
    "access_token": "***",
    "expires_in": 1209599,
    "refresh_token": "***",
    "refresh_token_expires_in": 7775954,
    "token_type": "Bearer",
    "scope": "spark:kms ..."
  }
```

Using Service App with wxc_sdk

Read secrets from environment

```
SERVICE_APP_CLIENT_ID=Cc96b6bfd15c0e75d3f
SERVICE_APP_CLIENT_SECRET=d602c66bd8cd32a
SERVICE_APP_REFRESH_TOKEN=ZGI1ZTE5NDEtNzY
```

Create access token and persist tokens in YML file

```
access_token: NTU5NTYxNjItZjU
expires_at: 2024-02-06 15:17:29.083739+00:00
expires_in: 1209599
refresh_token: ZGI1ZTE5NDEtNz
refresh_token_expires_at: 2024-04-22 15:17:29.083739+00:00
refresh_token_expires_in: 7775999
scope: spark-admin:workspaces_write Identity:one_time_password identity:placeonetimepassword_create
  spark:people_read identity:tokens_write spark-admin:workspace_locations_read spark-admin:workspaces_read
  spark:devices_write spark:devices_read spark:kms spark-admin:devices_read spark-admin:workspace_locations_write
  identity:tokens_read spark-admin:licenses_read spark-admin:telephony_config_read
  spark-admin:telephony_config_write spark-admin:devices_write spark-admin:people_read
```

Use tokens to call endpoints

https://github.com/jeokrohn/wxc_sdk/blob/master/examples/service_app.py

Using Service App with wxc_sdk

Read secrets from environment

```
SERVICE_APP_CLIENT_ID=Cc96b6bfd15c0e75d3f
SERVICE_APP_CLIENT_SECRET=d602c66bd8cd32a
SERVICE_APP_REFRESH_TOKEN=ZGI1ZTE5NDEtNzY
```

Create access token and persis

```
access_token: NTU5NTYxNjItZjU
expires_at: 2024-02-06 15:17:29.083739+00:00
expires_in: 1209599
refresh_token: ZGI1ZTE5NDEtNz
refresh_token_expires_at: 2024-04-22 15:17:29.083739+00:00
refresh_token_expires_in: 7775999
scope: spark-admin:workspaces_write Identity:one_time_password identity:placeonet
  spark:people_read identity:tokens_write spark-admin:workspace_locations_read sp
  spark:devices_write spark:devices_read spark:kms spark-admin:devices_read spark
  identity:tokens_read spark-admin:licenses_read spark-admin:telephony_config_rea
  spark-admin:telephony_config_write spark-admin:devices_write spark-admin:people
```

Use tokens to call endpoints

```python
def get_access_token() -> Tokens:
    """
    Get a new access token using refresh token, service app client id, service app client secret
    """
    tokens = Tokens(refresh_token=getenv('SERVICE_APP_REFRESH_TOKEN'))
    integration = Integration(client_id=getenv('SERVICE_APP_CLIENT_ID'),
                              client_secret=getenv('SERVICE_APP_CLIENT_SECRET'),
                              scopes=[], redirect_url=None)
    integration.refresh(tokens=tokens)
    write_tokens_to_file(tokens)
    return tokens


def get_tokens() -> Optional[Tokens]:
    """
    Get tokens from cache or create new access token using service app credentials
    """
    # try to read from file
    tokens = read_tokens_from_file()
    # .. or create new access token using refresh token
    if tokens is None:
        tokens = get_access_token()
    if tokens.remaining < 24 * 60 * 60:
        tokens = get_access_token()
    return tokens
```

https://github.com/jeokrohn/wxc_sdk/blob/master/examples/service_app.py

Using Service App with wxc_sdk

Read secrets from environment

```
SERVICE_APP_CLIENT_ID=Cc96b6bfd15c0e75d3f
SERVICE_APP_CLIENT_SECRET=d602c66bd8cd32a
SERVICE_APP_REFRESH_TOKEN=ZGI1ZTE5NDEtNzY
```

Create access token and persist tokens in YML file

```
access_token: NTU5NTYxNjItZjU
expires_at: 2024-02-06 15:17:29.083739+00:00
expires_in: 1209599
refresh_token: ZGI1ZTE5NDEtNz
refresh_token_expires_at: 2024-04-22 15:17:29.083739+00:00
refresh_token_expires_in: 7775999
scope: spark-admin:workspaces_write Identity:one_time_password identity:placeonetimepassword_cr
  spark:people_read identity:tokens_write spark-admin:workspace_locations_read spark-admin:work
  spark:devices_write spark:devices_read spark:kms spark-admin:devices_read spark-admin:workspa
  identity:tokens_read spark-admin:licenses_read spark-admin:telephony_config_read
  spark-admin:telephony_config_write spark-admin:devices_write spark-admin:people_read
```

Use tokens to call endpoints

```python
# get tokens and dump to console
tokens = get_tokens()
print(dumps(loads(tokens.json()), indent=2))
print()
print('scopes:')
print('\n'.join(f' * {s}' for s in sorted(tokens.scope.split())))

# use tokens to access APIs
api = WebexSimpleApi(tokens=tokens)

users = list(api.people.list())
print(f'{len(users)} users')

queues = list(api.telephony.callqueue.list())
print(f'{len(queues)} call queues')
```

https://github.com/jeokrohn/wxc_sdk/blob/master/examples/service_app.py

Token Overview

	Developer Token	Integration Token	Service App Token
How to get	From developer.webex.com	Requires OAuth auth code authorization flow, web server required	Service app access to org granted by org admin. Service app owner creates token on developer.webex.com
Granular Access Control	No, always has all scopes	Set of scopes assigned to integration	Set of scopes assigned to service app
Actor	Owner of developer token	User granting authorization	Service app
Lifetime	12 hrs	Access token: 14 d Refresh token: 90 d (extended when getting new access token)	
Use cases	Development, Tests	(Web) applications acting on behalf of user	(Web) services explicitly authorized by org admin

Use Cases / Examples

Use Cases

- Scripts
  - Bulk Provisioning: user, calling features, ...
  - Validation: check/verify settings; scripting saves you from navigating through individual menus in ControlHub
  - Automation reduces the risk of errors in repetitive tasks
  - CLI tools
- Integration with existing enterprise mangement systems/tools
- Backend for web services (portal applications)

SDK Examples

- Examples available at:
  https://wxc-sdk.readthedocs.io/en/latest/examples.html
  https://github.com/jeokrohn/wxc_sdk/tree/master/examples

```
(wxc-sdk-NNVrdgRm-py3.9) jkrohn@JKROHN-M-942M examples % ./reset_call_forwarding.py
```

```python
api = WebexSimpleApi()

# get all calling users
start = time.perf_counter_ns()
calling_users = [user for user in api.people.list(calling_data=True)
                 if user.location_id]
print(f'Got {len(calling_users)} calling users in '
      f'{(time.perf_counter_ns() - start) / 1e6:.3f} ms')

# set call forwarding to default for all users
with ThreadPoolExecutor() as pool:
    # default call forwarding settings
    forwarding = PersonForwardingSetting.default()

    # schedule update for each user and wait for completion
    start = time.perf_counter_ns()
    list(pool.map(
        lambda user: api.person_settings.forwarding.configure(person_id=user.person_id,
                                                              forwarding=forwarding),
        calling_users))
    print(f'Reset call forwarding to default for {len(calling_users)} users in '
          f'{(time.perf_counter_ns() - start) / 1e6:.3f} ms')
```

Examples

- queue_helper.py: read/update call queue join state of users
- call_intercept.py: read/update call intercept settings of a user
- us_holidays_async.py: create schedule with US holidays for all US location
- reset_call_forwarding.py: reset call forwarding settings for all users
- users_wo_devices.py: identify users without devices
- catch_tns.py: pool unassigned TNs on hunt groups to catch calls to unassigned TNs
- room_devices.py: remove calling entitlements from selected Workspaces
- ...

Example: User Web Portal

User Web Portal

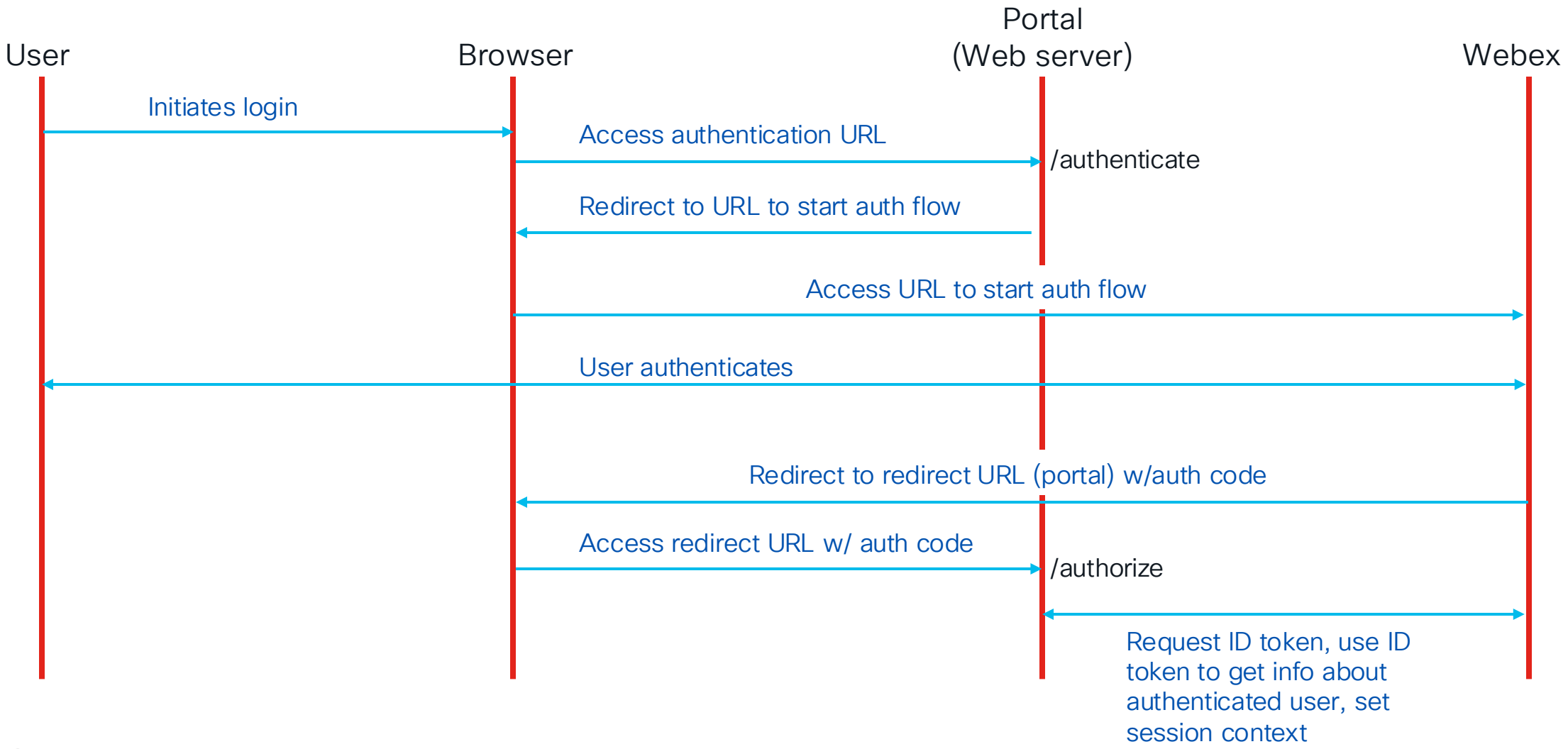
- Only a limited set of configuration options is available for users on settings.webex.com
- User portal can expose additional options (which usually require admin privileges)
- Perfect example for using service app tokens
  - User does not have the required privileges → integration token granted by user not sufficient

Concepts

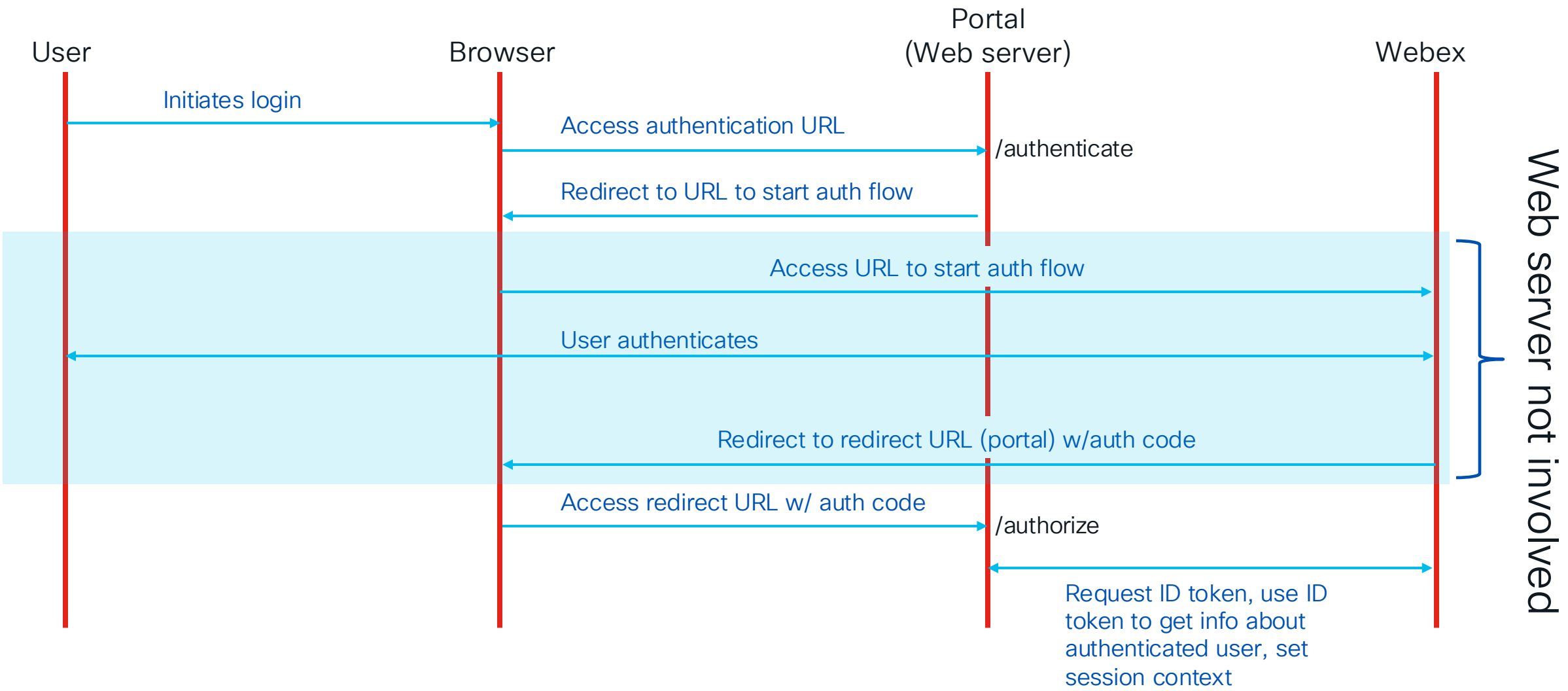
- Web based portal
- Authenticate users using “Login with Webex”
- For provisioning operations portal uses service app tokens
  - No tokens exposed in browser
- Browser only interacts with portal endpoints
  - Includes Javascript Ajax access
  - Portal acts as proxy between browser and Webex APIs
- APIs implemented using flask-restx → Swagger UI for free

https://developer.webex.com/create/docs/login-with-webex
https://github.com/jeokrohn/BRKCOL-3015

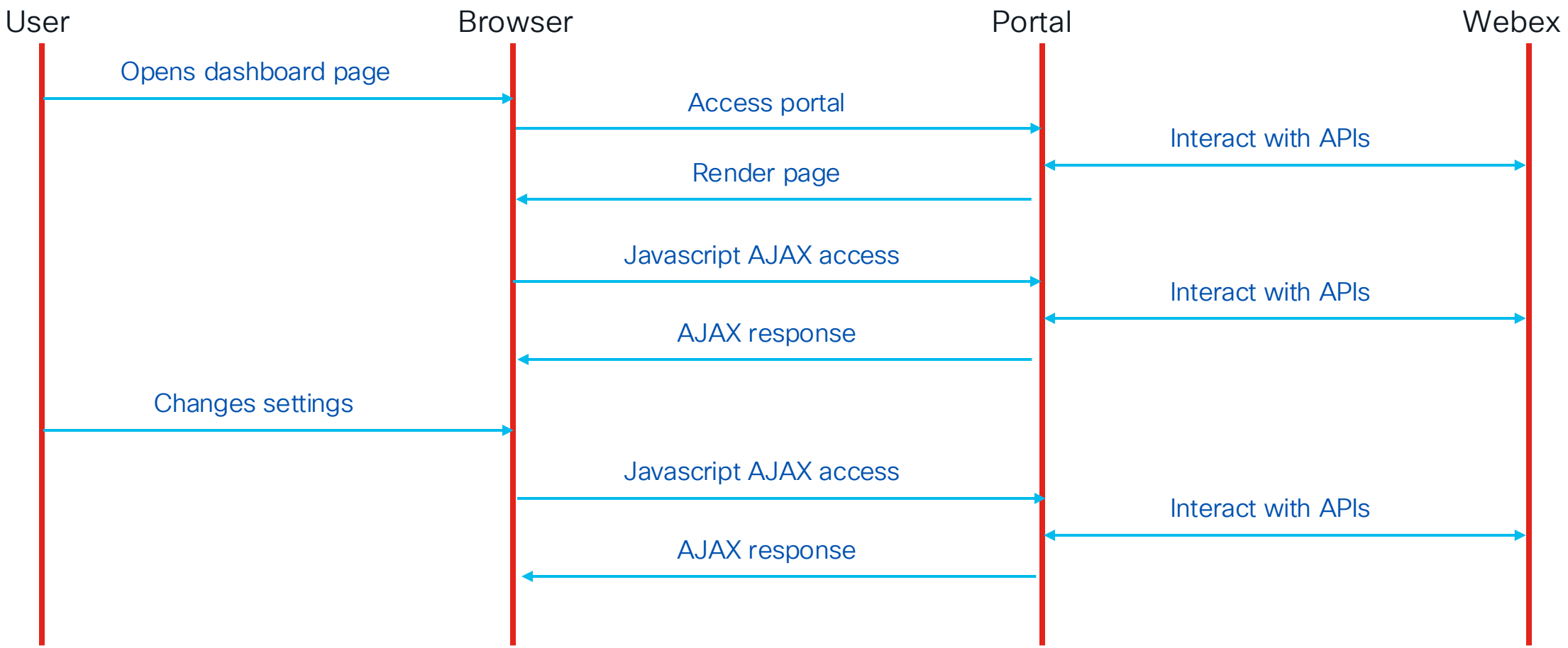
Login with Webex



Login with Webex



Portal Transactions



Rendering Data

index.html

```
<!-- phones -->
<div class="row">
  <div class="col-xl-8 col-lg-7 mb-4" id="phoneCard">
    <div class="card  shadow mb-4 h-100">
      <div class="card-header py-3">Phones</div>
      <div class="card-body">
        <div class="table-responsive">
          <table class="table table-striped table-bordered dt-responsive nowrap" id="userPhones">
          <!-- the table is empty and will be set in $(document).ready -->
          </table>
        </div>
        <div class="row">
          <div class="text-xs" id="status">
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

index.html

```
{% block scripts %}
  <script type="text/javascript">
    $(document).ready(function () {
      ...
      // set up DataTable for phones
      $('#userPhones').DataTable({
        searching: false,
        data: [],
        columns: [
          {title: "Model", data: "model"},
          {title: "Mac", data: "mac"},
          {title: "Status", data: "status"},
        ]
      });
      update_user_phones()
```

Rendering Data

user_phones.js

```
function get_user_phones() {
   return $.ajax({type: 'GET', url: '/userphones'})
}

function update_user_phones() {
   const card = document.getElementById("phoneCard")
   const status = card.querySelector("#status")
   status.textContent = "...getting phone information"

   // get phones from server and update table
   get_user_phones().then(function (data) {
      console.log(data)

      if (data["success"] == false) {
         status.textContent = "Error: " + data["message"]
      } else {
         status.textContent = ""
         $('#userPhones').DataTable().clear()
         $('#userPhones').DataTable().rows.add(data['rows'])
         $('#userPhones').DataTable().columns.adjust().draw()
      }
   })
}
```

index.html

```
{% block scripts %}
   <script type="text/javascript">
      $(document).ready(function () {
         ...
         // set up DataTable for phones
         $('#userPhones').DataTable({
            searching: false,
            data: [],
            columns: [
               {title: "Model", data: "model"},
               {title: "Mac", data: "mac"},
               {title: "Status", data: "status"},
            ]
         });
         update_user_phones()
```

es">

Rendering Data

user_phones.js

```
function get_user_phones() {
   return $.ajax({type: 'GET', url: ‘api/userphones'})
}

function update_user_phones() {
   const card = document.getElementById("phoneCard")
   const status = card.querySelector("#status")
   status.textContent = "...getting phone information"

   // get phones from server and update table
   get_user_phones().then(function (data) {
      console.log(data)

      if (data["success"] == false) {
         status.textContent = "Error: " + data["message"]
      } else {
         status.textContent = ""
         $('#userPhones').DataTable().clear()
         $('#userPhones').DataTable().rows.add(data['rows'])
         $('#userPhones').DataTable().columns.adjust().draw()
      }
   })
}
```

api/__init__.py

```
@api.route('/userphones')
class UserPhones(Resource):
   """
   Get phones of current user
   """

   @staticmethod
   @assert_user
   def get():
      """
      Get phones of current user.
      Returns a JSON object with:
         * success: True if the operation was successful
         * rows: List of phone data rows. Each row in the table will contain:
            * product: Product name of the phone
            * mac: MAC address of the phone in colon-separated format
            * connection_status: Connection status of the phone
      """

      def mac_with_colons(mac: str) -> str:
         octets = (mac[i:i + 2] for i in range(0, len(mac), 2))
         return ':'.join(octets)

      user = session.get('user')
      user: Person
      ca: AppWithTokens = current_app
      path = urlparse(request.url).path
      try:
         log.debug(f'"{path}": getting user phones')
         devices = list(ca.api.devices.list(person_id=user.person_id))
      except RestError as e:
         log.error(f'"{path}": getting user phones failed: {e}')
         return {'success': False,
               'message': f'{e}'}
      log.debug(f'"{path}": returning device data')
      return {'success': True,
            'rows': [{'model': device.product,
                  'mac': mac_with_colons(device.mac),
                  'status': device.connection_status}
                 for device in devices
                 if device.product_type == ProductType.phone]}
```

Demo Web App

Web App

- Run app in docker container
  - docker-compose up –d –build -build to re-build the image
  - docker-compose logs –f show logs
- Run web_app/app.py
- Portal URL: http://localhost:5010
- Swagger UI: http://localhost:5010/api/docs
  - Review/test the API endpoints
  - User must be authenticated

Disclaimer, Where to Go Next

- Demo code is not “production ready”
  - Missing token lifetime monitoring
  - Filesysten based session backend
  - ...
- Where to go next
  - Add roles: can be based on groups in Webex
  - Additional functions: manage devices, bulk update, ...

Closing

References

- Webex for Developers: https://developer.webex.com/
- Examples for the session on GitHub https://github.com/jeokrohn/BRKCOL-3015
- Python SDK: https://pypi.org/project/wxc-sdk/
- SDK Examples available at: https://wxc-sdk.readthedocs.io/en/latest/examples.html https://github.com/jeokrohn/wxc_sdk/tree/master/examples
- Call control bot: https://wxc-callcontrol.readthedocs.io/en/latest/
- Other repositories: https://github.com/jeokrohn?tab=repositories

webexone 25

Share your experience!

Scan the QR code for the post-session survey


webexone 25