Using Web Hooks from inside the Firewall

Johannes Krohn

Technical Marketing Engineer

10. Sep 2019

Agenda

Code, examples available at https://github.com/jeokrohn/duwebhook

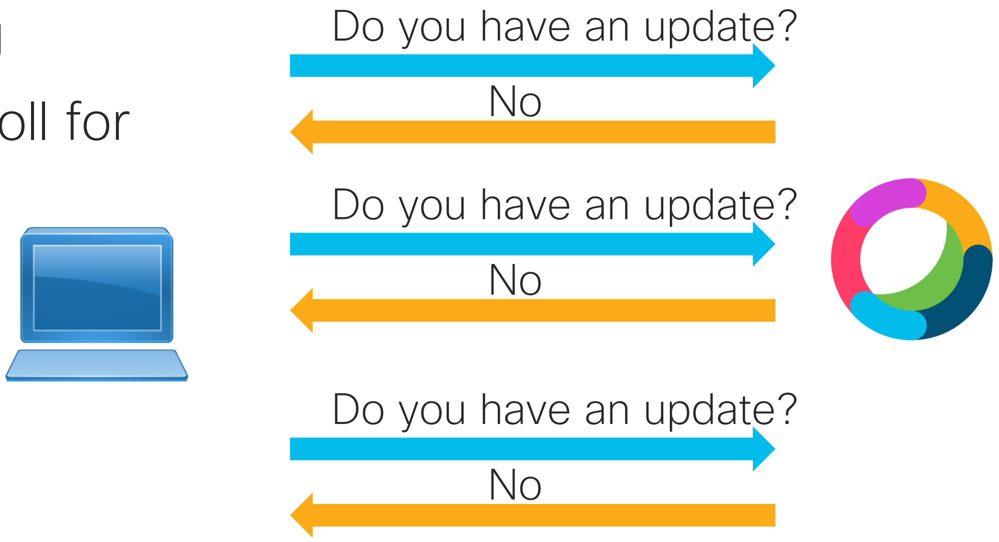
- Webhooks
- Integrations vs. Bots
- Building a Basic Bot
- Dirty Hack

Webhooks

Webhooks – Problem Statement

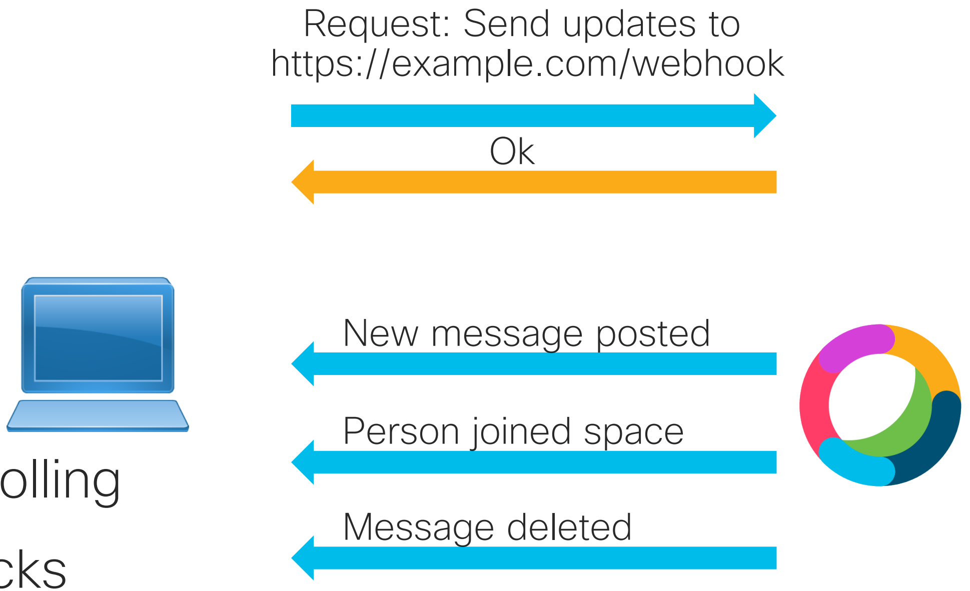
- Polling for events is inefficient and does not scale
- Too many instances polling
- Too many event types to poll for

→ not really 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

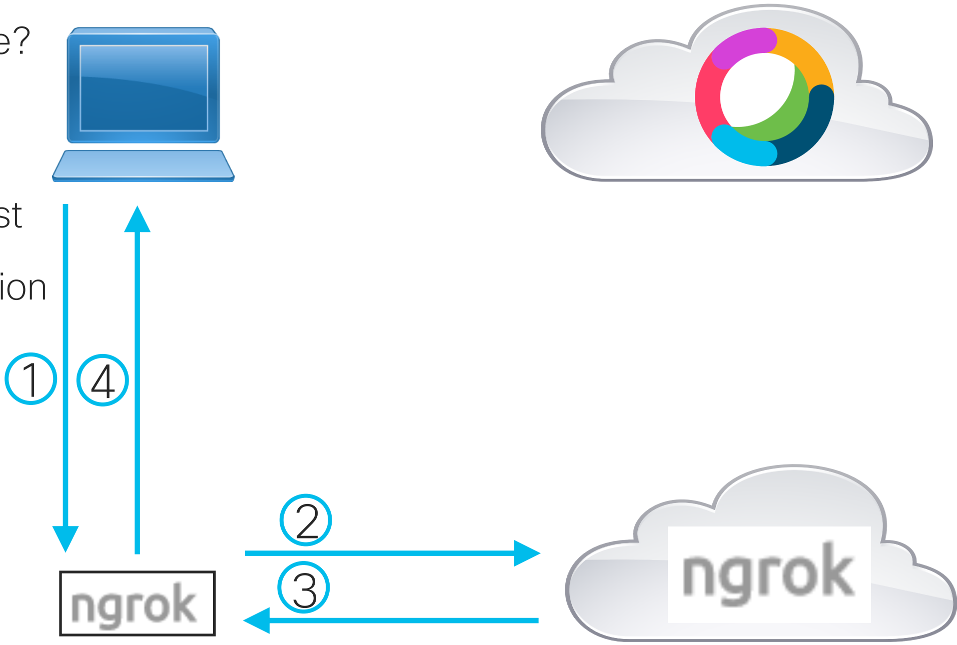
- What if code runs on host not publicly reachable?
  - Inside firewall
  - No public hostname
- Ngrok: cloud service to tunnel public URL to host
- Ngrok client on host creates persistent connection
- Ngrok client on host relays requests received from the cloud to localhost

① Start ngrok client

② Create persistent connection

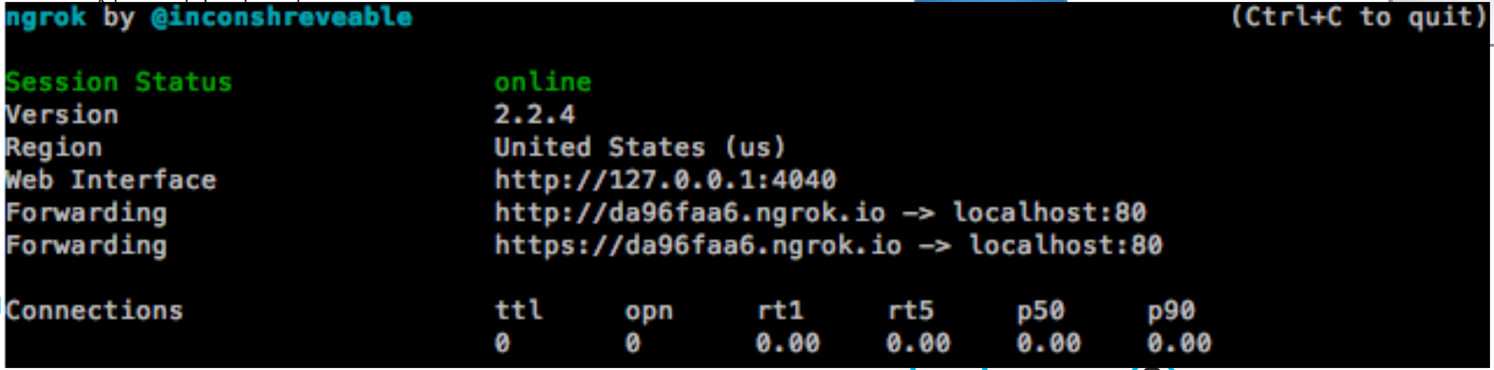
③ Obtain public URL

④ Report public URL



Webhooks w/o Public URI

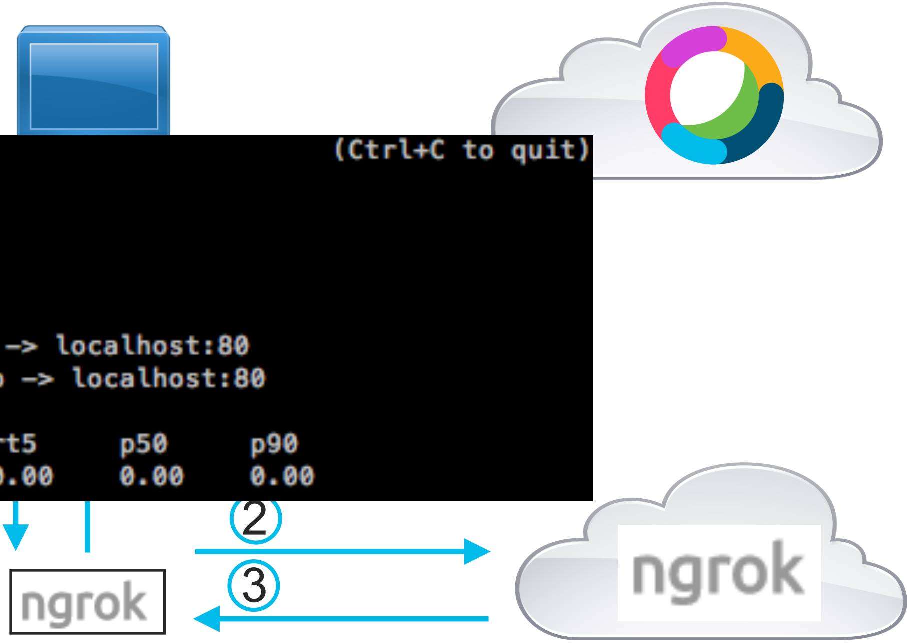
- What if code runs on host not publicly reachable?
  - Inside firewall



```
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

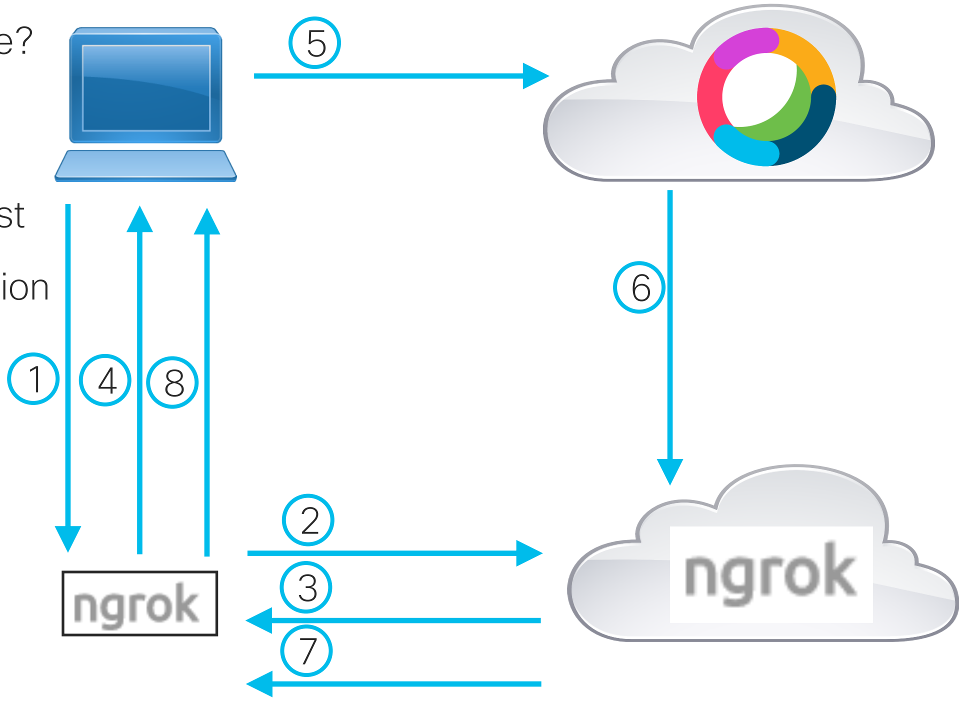
3 Obtain public URL

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



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 Webex Teams conversations
- Typical types of bots:
  - Notifier: post notifications to Webex Teams 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 authorisation
  - User approves authorisation 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.

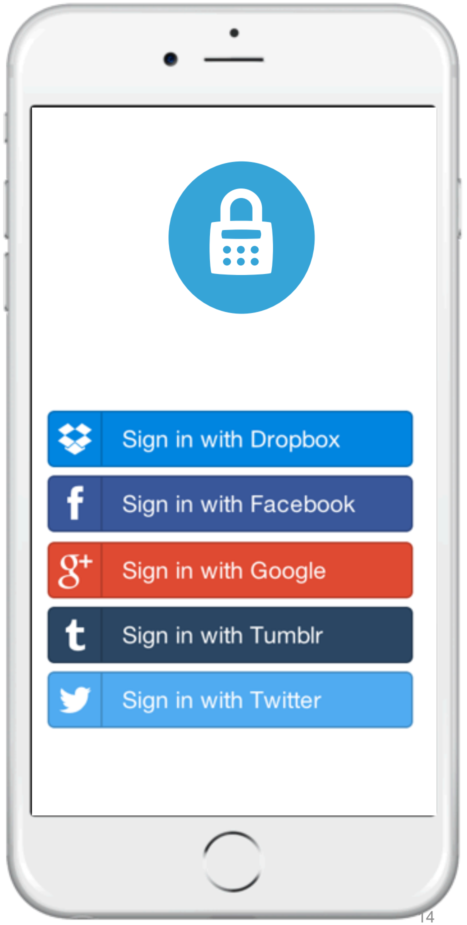
Integrations: Secure with oAuth

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

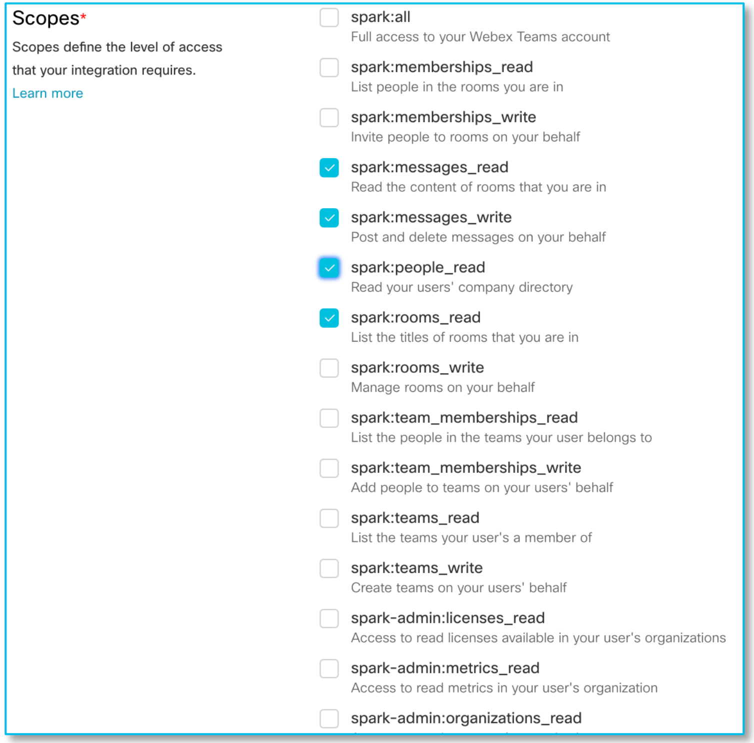
A personal access token will make calls on your behalf, but in production, you will need your app to post on behalf of others.

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 Authorisation code for an Access Token
- Use this Access Token to make your API calls



Access Level of integration



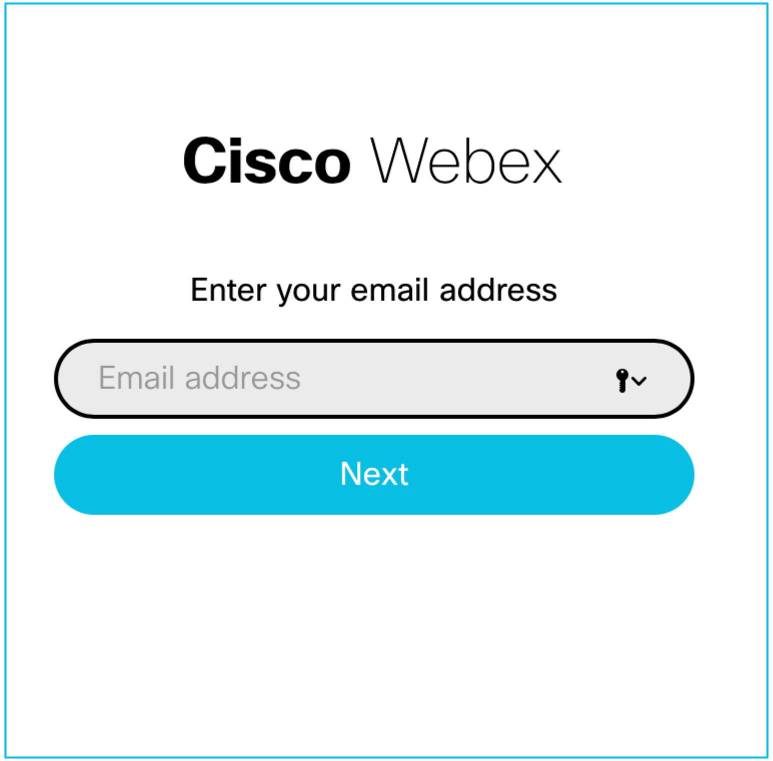
Scopes*

Scopes define the level of access that your integration requires.

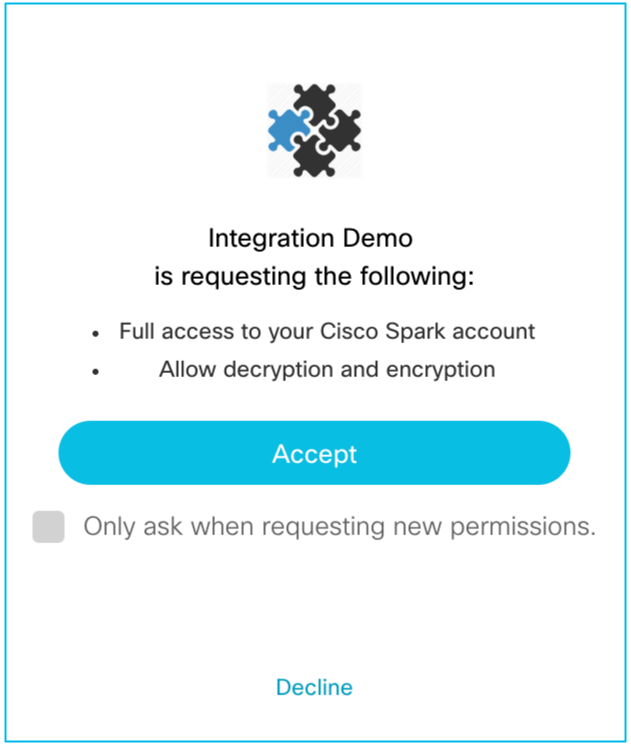
Learn more

- [ ] 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
- [✓] spark:messages_read
  Read the content of rooms that you are in
- [✓] spark:messages_write
  Post and delete messages on your behalf
- [✓] spark:people_read
  Read your users' company directory
- [✓] 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


Integration Demo
is requesting the following:

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

oAuth Authorization Code Flow Summary

1. Application Requests *auth code*

Browser redirect to Spark Authentication

2. Webex returns the *auth code* to application

Browser redirect to Application

3. Request an *access token*

HTTP GET request to Webex Teams API

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

HTTP GET response from Webex Teams API

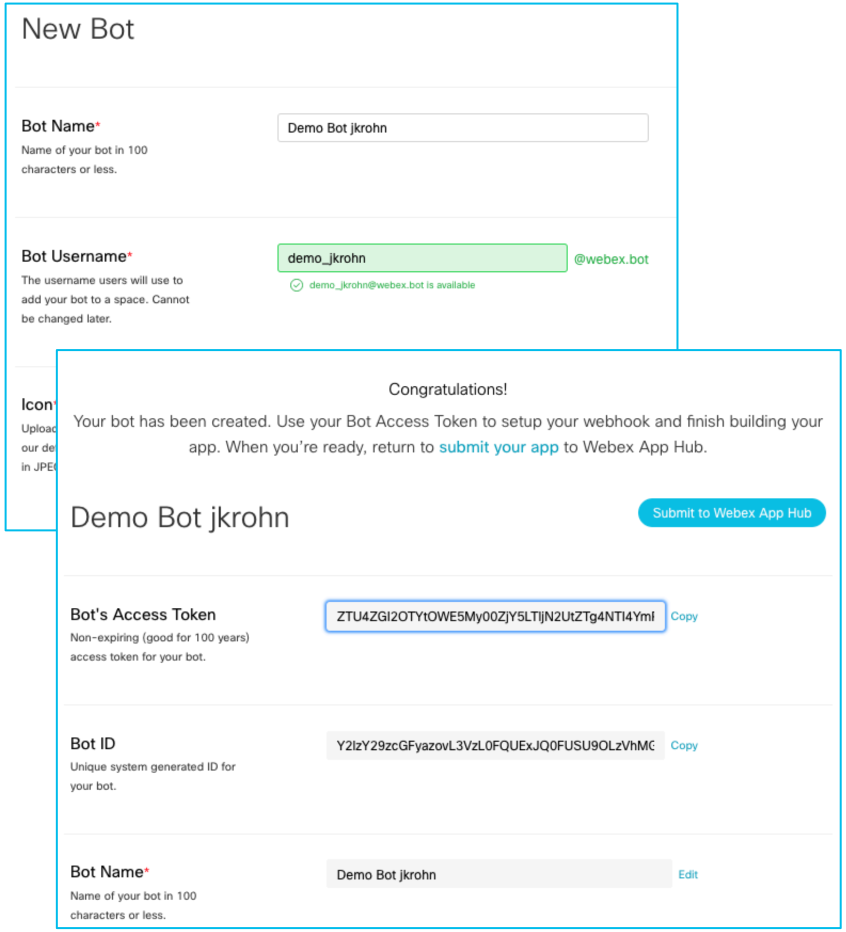
Deploying an Integration

- 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?

Building a Basic Bot

Creating the Bot

- developer.webex.com
- Obtain:
  - Bot email
  - Bot ID
  - Bot access token (only shown once!!)



New Bot

Bot Name*
Name of your bot in 100 characters or less.
Demo Bot jkrohn

Bot Username*
The username users will use to add your bot to a space. Cannot be changed later.
demo_jkrohn @webex.bot
demo_jkrohn@webex.bot is available

Icon
Upload
our de
in JPE

Congratulations!

Your bot has been created. Use your Bot Access Token to setup your webhook and finish building your app. When you're ready, return to submit your app to Webex App Hub.

Demo Bot jkrohn

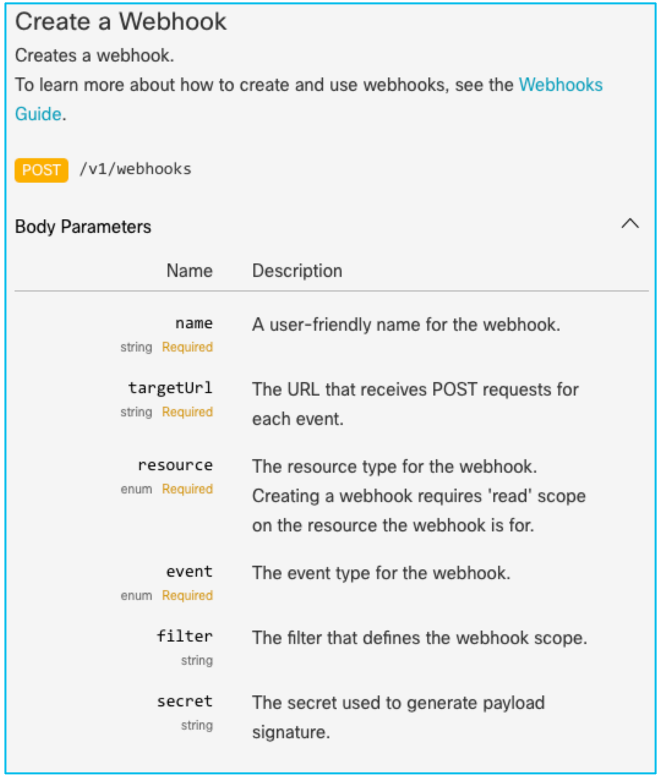
Bot's Access Token
Non-expiring (good for 100 years) access token for your bot.
ZTU4ZGI2OTYtOWE5My00ZjY5LTljN2UtZTg4NTI4YmI

Bot ID
Unique system generated ID for your bot.
Y2lzY29zcGFyazovL3VzL0FQUExJQ0FUSU9OLzVhMG

Bot Name*
Name of your bot in 100 characters or less.
Demo Bot jkrohn

Receiving Notifications

- Webhook is an HTTP callback
- When creating a notification Webhook an absolute target URL has to be provided
- Again: when running inside the firewall we typically cannot provide that URL



Create a Webhook

Creates a webhook.

To learn more about how to create and use webhooks, see the Webhooks Guide.

POST `/v1/webhooks`

Body Parameters

Name	Description
`name` string Required	A user-friendly name for the webhook.
`targetUrl` string Required	The URL that receives POST requests for each event.
`resource` enum Required	The resource type for the webhook. Creating a webhook requires 'read' scope on the resource the webhook is for.
`event` enum Required	The event type for the webhook.
`filter` string	The filter that defines the webhook scope.
`secret` string	The secret used to generate payload signature.

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

Demo: basic bot

Ingredients

- `threading`: Thread to start and monitor an Ngrok process
- `subprocess`: running ngrok locally
- `webexteamssdk`: driving the Webex Teams API
- `webexteamsbot`: simple bot framework
- `beautifulsoup4`: parsing of web pages
- `requests_toolbelt`: multipart mime message creation

Dirty Hack

Device Registration

- Webex Teams Devices (and apps) register with a registration service in the cloud
  - REST endpoint: https://wdm-a.wbx2.com/wdm/api/v1/devices
- Websocket URI obtained during registration
- Websocket is used for any type of notification
  - Message activity
  - Communication with Key Management Server (KMS)
  - ...

Encryption

- Message content received via the Websocket is encrypted
- Getting keys from KMS is too complex .. although possible
- For messaging all we need is the message ID and can obtain the message in the clear via the public Webex Teams apis
-

Message ID Format

- Message IDs on the Websocket are UUIDs
  - example: ‘946e4f40-d002-11e9-9ccb-a7e5bdebafb3’
- Public APIs typically expect a different ID format
  - Example: ‘Y2lzY29zcGFyazovL3VzL01FU1NBR0UvOTQ2ZTRmNDAtZDAwMi0xMWU5LTljY2ItYTdlNWJkZWJhZmIz’
  - This can be base64 decoded to: ‘ciscospark://us/MESSAGE/946e4f40-d002-11e9-9ccb-a7e5bdebafb3‘
- Interestingly there is no need to map from UUID to Webex ID
- .. b/c the public APIs also accept UUIDs ☺

Additional Material

References

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