API Call Introduction

Python, Ansible, & cURL

2023/05/26

Jeremy White, Cristian Sîrbu

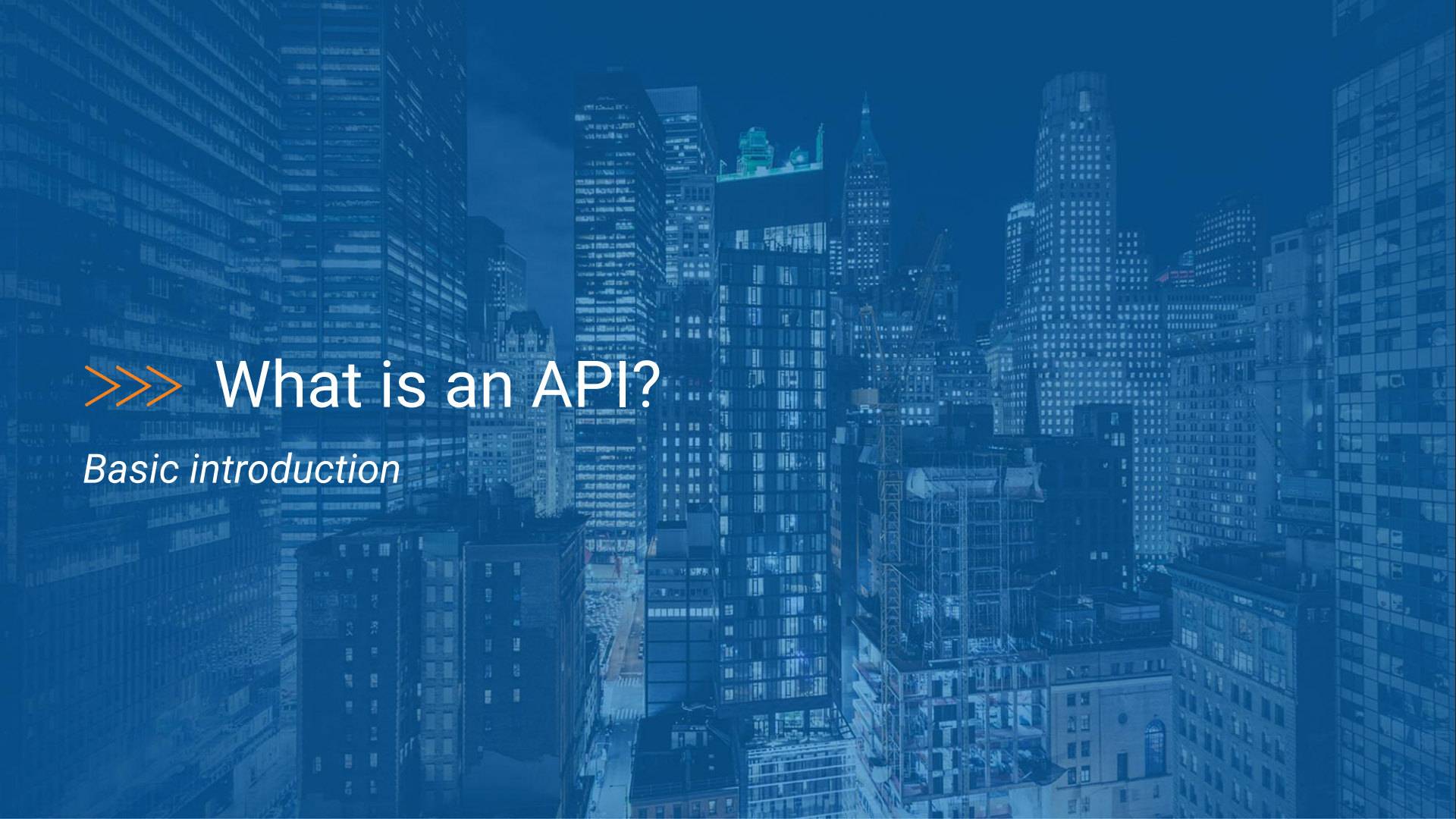
Agenda

What is an API?

Python

Ansible

cURL



What is an API?

Basic introduction

>>> What is an API?

The industry is transitioning from CLI to an API first model

- CLI is for humans
- APIs are for machine to machine communication
- APIs do not replace CLI but serve a different purpose
- APIs can have a profound impact on operations
- APIs facilitate operational efficiency

>>> What is an API?

Authentication

- Basic Authentication
  - Common in User/Pass authentication, each request passes the authentication token for access
  - Authorization: Basic <Base64 encoded user & pass>
- Token Authentication
  - Implementation can be less standardized and can be user or system defined tokens
  - Authorization: Token <user specific token>
- OAuth2 Bearer Token Authentication
  - Server generated token, typically bound to a user & application.
  - Authorization: Bearer <token>
- Session Authentication
  - Uses existing authentication to create a session and subsequent calls are performed with the existing session until the session is ended or expires. Dependent on Cookies

>>> What is an API?

Methods - Remember CRUD

- **POST**
  - Create a single item /api/users/ - 201
- **GET**
  - Retrieve a collection /api/users/ - 200
  - Retrieve a single item /api/users/{id}/ - 200
- **PUT**
  - Replace a single item /api/users/{id}/ - 200
- **PATCH**
  - Update a single item /api/users/{id}/ - 200
- **DELETE**
  - Delete a single item /api/users/{id}/ - 204


Python
Requests

>>> Python

Requests

- The Requests library is one of the simplest libraries to use when interacting with APIs.
- This building blocks of several popular SDK libraries.

```python
import requests

url = "https://example.com/api/users/"
session = requests.Session()
session.auth=("user", "somepass")

# Session Authentication
resp = requests.get(url)
```

```python
import requests

url = "https://example.com/api/users/"

# No Authentication
resp = requests.get(url)

# Basic Authentication
resp = requests.get(url, auth=("user", "somepass"))

# Token Authentication
resp = requests.get(
    url,
    headers={
        "Authorization": "Token ABC123"
    }
)

# Bearer Token Authentication
resp = requests.get(
    url,
    headers={
        "Authorization": "Bearer ABC123"
    }
)
```

>>> Python

Requests

- Providing and interacting with data via XML & JSON are common data formats
- Some APIs only support one or the other data type
- Some APIs could expect a specific headers
- Requests will automatically add Content-Type: Application/JSON to the headers when JSON attribute is used.

```
# JSON POST Authentication With Headers
resp = requests.post(
    url,
    headers={
        "content-type": "application/json"
    },
    data={"var1": "var value"}
)
# JSON POST Authentication With Automatic Headers
resp = requests.post(
    url,
    json={"var1": "var value"}
)
```

>>> Python

Requests

```python
# XML POST With Headers
data = "<?xml version='1.0' encoding='utf-8'?><var1>var value</var1>"
resp = requests.post(
    url,
    headers={
        "content-type": "application/xml"
    },
    data=data
)
```

>>> Python

Requests

- Parsing JSON from the response is as simple a .json()
- XML is best worked with as an XML Element Tree parsed from the response .content

```
# JSON Response
resp = requests.get(url)
resp.json()
{'var1': 'var value'}

# XML Response
resp = requests.get(url)
from xml.etree import ElementTree
ElementTree.fromstring(resp.content)
"<?xml version='1.0' encoding='utf-8'?><var1>var value</var1>"
```

>>> Python

Requests

- Status checking can be done at the individual status level OR the response provides the .ok attribute that will count all 2XX response as successful
- Sometimes checking the exact response code may be required via the .status_code attribute

```
# HTTP Response Code
resp = requests.get(url)
resp.status_code
200
resp.status_code == 201
False

# HTTP Response OK
resp = requests.get(url)
resp.ok
True
```


Ansible
URI Module

>>> Ansible

URI Module

- Although this is a built in module for Ansible, URI does NOT support any idempotent tasks, and only the GET method can be run in check_mode
- URI expects a status_code of 200 OR you can provide a list of appropriate status codes.

```yaml
- name: "EXAMPLE 1"
  ansible.builtin.uri:
    url: https://example.com/api/users/
    method: GET
    force_basic_auth: yes
    username: some_user
    password: some_pass
    status_code:
      - 200

- name: "EXAMPLE 2"
  ansible.builtin.uri:
    url: https://example.com/api/users/
    method: POST
    force_basic_auth: yes
    username: some_user
    password: some_pass
    body_format: json
    body:
      var1: var value1
      var2: var value2
    status_code:
      - 201
      - 204
```

>>> Ansible

URI

- The response must be registered to use in subsequent tasks
- The registered response is the response object NOT just the response payload (similar to Python but commonly mistakenly understood)
- Ansible pairs perfectly with structured data

```yaml
- name: "EXAMPLE 3"
  ansible.builtin.uri:
    url: https://example.com/api/users/
    method: GET
    force_basic_auth: yes
    username: some_user
    password: some_pass
    status_code:
      - 200
    return_content: yes
    body_format: json
  register: output

- name: "EXAMPLE 4"
  ansible.builtin.debug:
    msg: "{{ output.json"
```

>>> Ansible

URI

- Headers become even easier
- URI makes API calls not as scary once you get the basics
- Ansible XML task and filters are best for interacting with XML via Ansible
- Parsing XML to a Dictionary can become even more tedious to troubleshoot when done in Ansible

```yaml
- name: "EXAMPLE 5"
  ansible.builtin.uri:
    url: https://example.com/api/users/
    method: GET
    status_code:
      - 200
    headers:
      content-type: application/json
      accept: application/json
      Authorization: Token ABC123
```


cURL
Simple Tried & True

>>> cURL

Simple Tried & True

- Can be low to no frills command line hero
- Relies heavily on positional arguments
- Phenomenal tool for quick communication litmus test on Linux machines (most ship with cURL)

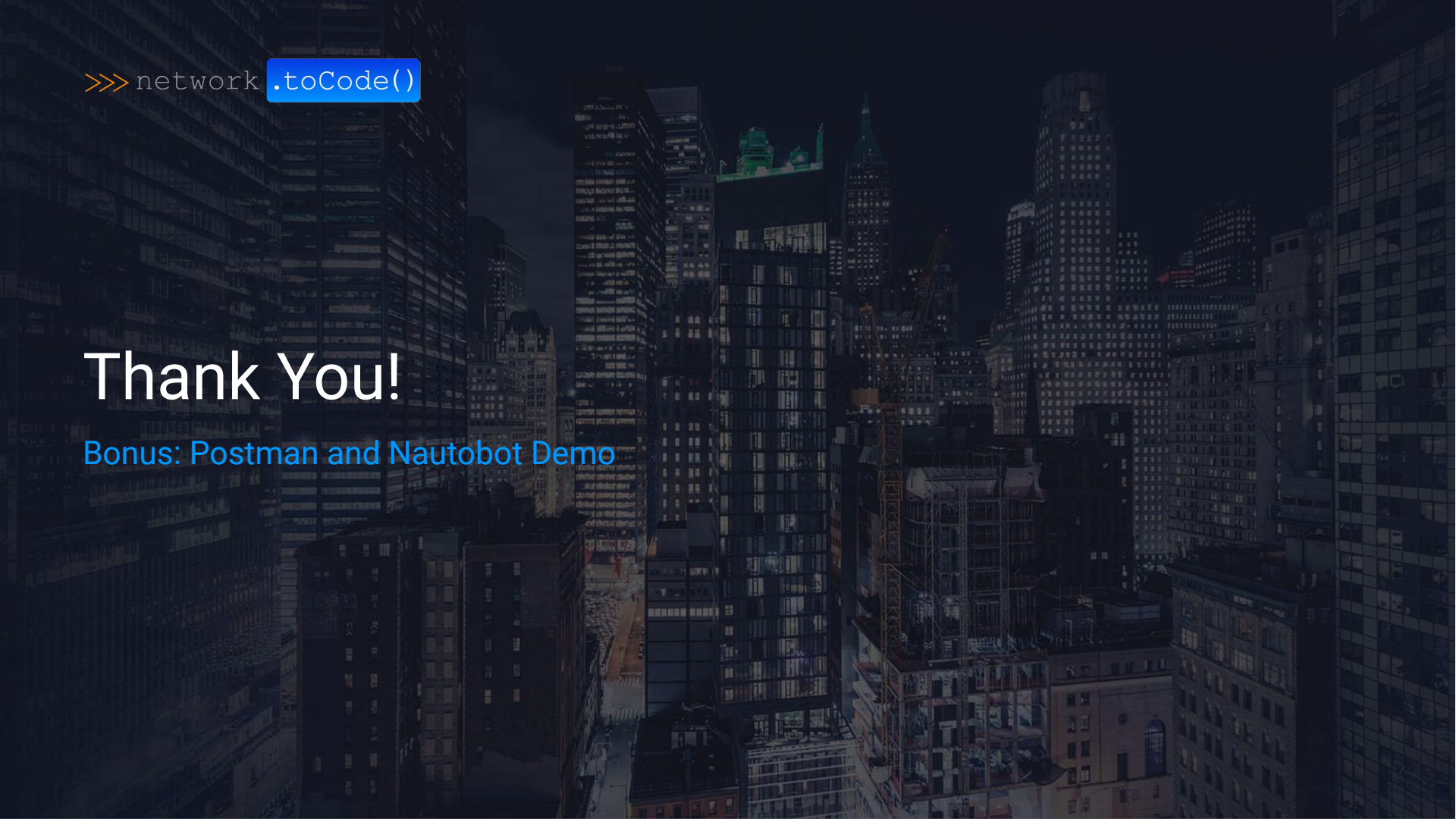
```
~ # Simple HTTP GET without any arguments or auth
~ curl https://example.com
~
~ # Adding Authentication Headers
~ curl -H "Accept: application/json" https://example.com
~
~ # Sending output to a file
~ curl https://example.com > some.json
```

>>> cURL

Simple Tried & True

The more you add to your commands the harder it is to understand.

```
~ curl -X POST https://example.com/api/users/
~
~ # POST with no data
~ curl https://example.com > some.json
~
~
~ # POST with no data
~ curl -X POST https://example.com/api/users/
~
~ # POST with data
~ curl -X POST -d '{"somevar": "some value"}' https://example.com/api/users/
~
~ # POST with data from file
~ curl -X POST https://example.com/api/users/ -d @some.json
```


>>> network .toCode()
Thank You!
Bonus: Postman and Nautobot Demo