Getting large data sets with the Zendesk API and Python

ON THIS PAGE

A simple script like the one you can write in the basic Python tutorial called Making requests to the Ticketing API is fine for getting up to two dozen or so records from your Zendesk product. However, to retrieve several hundred or several thousand records, a script has to perform the following tasks:

- Make the basic request
- Paginate through all the results
- Guard against the rate limit
- Sideload related data, if applicable
- Serialize the data, if you need to reuse it

This article shows you how to write a Python script that can retrieve large data sets with the Zendesk API. To run the examples, you'll need Python 3 and the Requests library.

After getting a large data set from the API, you might want to move it to a Microsoft Excel worksheet to more easily view and analyze the data. To learn how, see Write large data sets to Excel with Python and pandas.

For all the possible data you can retrieve from your Zendesk product, see the "JSON Format" tables of the Support and the Help Center API docs. Most APIs have a "List" endpoint for getting multiple records.

Disclaimer: Zendesk provides this article for instructional purposes only. Zendesk does not support or guarantee the code. Zendesk also can't provide support for third-party technologies such as Python.

Make the basic request

Suppose you want to download the four thousand posts in a community topic in your help center. Start with the basic request. Create a file named **list_posts.py** and paste the following code in it:

```
import requests
import os

# In production, store the API token in environment variables for security
ZENDESK_API_TOKEN = os.getenv('ZENDESK_API_TOKEN')
ZENDESK_USER_EMAIL = os.getenv('ZENDESK_EMAIL')
ZENDESK_SUBDOMAIN = 'YOUR_ZENDESK_SUBDOMAIN'

topic_id = 123456
topic_posts = []

url = f"https://{ZENDESK_SUBDOMAIN}.zendesk.com/api/v2/community/topics/{topic_id}/posts.json"

auth = f'{ZENDESK_USER_EMAIL}/token', ZENDESK_API_TOKEN

response = requests.get(url, auth=auth)

if response.status_code != 200:
    print(f'Error with status code {response.status_code}')
    exit()
data = response.json()
topic_posts.extend(data['posts'])  # Extend the empty list with the 'posts' data


for post in topic_posts:
    print(post['title'])
```

The general logic of the script is explained in Getting data from your Zendesk product in the basic Python tutorial.

Replace the value of **topic_id** with the id of a community topic in your help center.

Save the file. In your command line tool, navigate to the folder with the script and run the following command:

```
python3 list_posts.py
```

The response should return the first 30 posts in the community topic you specified.

Paginate through all the results

For bandwidth reasons, the API doesn't return large record sets all at once. Use the `page[size]` parameter in the request parameter to specify the number of items to return per page. Most endpoints limit this to a maximum of 100.

To capture all the records, create a while loop, stash the page data incrementally in a variable, and continue paginating until the `has_more` property nested in the `meta` JSON object is false. This indicates there are no further records. Add the highlighted lines to your script to do this:

```
import requests
import os

# In production, store credentials in environment variables
ZENDESK_API_TOKEN = os.getenv('ZENDESK_API_TOKEN')
ZENDESK_USER_EMAIL = os.getenv('ZENDESK_EMAIL')
ZENDESK_SUBDOMAIN = 'YOUR_ZENDESK_SUBDOMAIN'

topic_id = 123456
topic_posts = []

url = f"https://{ZENDESK_SUBDOMAIN}.zendesk.com/api/v2/community/topics/{topic_id}/posts.json?page[size]=100"

auth = f'{ZENDESK_USER_EMAIL}/token', ZENDESK_API_TOKEN

response = requests.get(url, auth=auth)
while url:
    if response.status_code != 200:
        print(f'Error with status code {response.status_code}')
        exit()
    data = response.json()
    topic_posts.extend(data['posts'])

    if data['meta']['has_more']:
        url = data['links']['next']
    else:
        url = None

for post in topic_posts:
    print(post['title'])
```

For an explanation of the logic, see Paginating through lists using cursor pagination.

Guard against the rate limit

If you make a lot of API requests in a short time, such as when paginating through a large data set, you might bump into the Zendesk API rate limit. The API stops processing any more requests until a certain amount of time has passed. For more information, see Usage limits in the API reference docs.

When you reach the rate limit, the API responds with a HTTP 429 Too Many Requests response code. The response has a `Retry-After` header that tells you how many seconds to wait before retrying.

Update the script with the highlighted lines to check for a 429 status code and wait if it's detected:

```
import time
import requests
import os

# In production, store credentials in environment variables
ZENDESK_API_TOKEN = os.getenv('ZENDESK_API_TOKEN')
ZENDESK_USER_EMAIL = os.getenv('ZENDESK_EMAIL')
ZENDESK_SUBDOMAIN = 'YOUR_ZENDESK_SUBDOMAIN'

topic_id = 123456
topic_posts = []

url = f"https://{ZENDESK_SUBDOMAIN}.zendesk.com/api/v2/community/topics/{topic_id}/posts.json?page[size]=100"

auth = f'{ZENDESK_USER_EMAIL}/token', ZENDESK_API_TOKEN

while url:
    response = requests.get(url, auth=auth)
    if response.status_code == 429:
        print('Rate limited! Please wait.')
        time.sleep(int(response.headers['retry-after']))
        continue
    if response.status_code != 200:
        print(f'Error with status code {response.status_code}')
        exit()
    data = response.json()
    topic_posts.extend(data['posts'])

    if data['meta']['has_more']:
        url = data['links']['next']
    else:
        url = None

for post in topic_posts:
    print(post['title'])
```

For more information, see Best practices for avoiding rate limiting.

Sideload related data

Suppose you want to display the author of each community post. The records returned by the posts API identify authors only by their Zendesk Support user id, not by their actual names. Example: `"author_id": 21436587`.

You could call the users API to get the name associated with each user id. However, this means calling the API for each post in your data set, potentially amounting to thousands of API calls.

A more efficient solution is to sideload the user records with the post records. Sideloading gets both recordsets in a single request. For more information, see Sideloading related records.

Update the script as follows (new lines highlighted) to sideload the users who authored the posts. Make sure to scroll horizontally to see the modified `url` variable.

```
import time
import requests

ZENDESK_API_TOKEN = os.getenv('ZENDESK_API_TOKEN')
ZENDESK_USER_EMAIL = os.getenv('ZENDESK_EMAIL')
ZENDESK_SUBDOMAIN = 'YOUR_ZENDESK_SUBDOMAIN'

topic_id = 123456
topic_posts = []
user_list = []
url = f"https://{ZENDESK_SUBDOMAIN}.zendesk.com/api/v2/community/topics/{topic_id}/posts.json?page[size]=100&include=users"

auth = f'{ZENDESK_USER_EMAIL}/token', ZENDESK_API_TOKEN

while url:
    response = requests.get(url, auth=auth)
    if response.status_code == 429:
        print('Rate limited! Please wait.')
        time.sleep(int(response.headers['retry-after']))
        continue
    if response.status_code != 200:
        print(f'Error with status code {response.status_code}')
        exit()
    data = response.json()
    topic_posts.extend(data['posts'])
    user_list.extend(data['users'])

    if data['meta']['has_more']:
        url = data['links']['next']
```

```
30        else:
31            url = None
32
33    for post in topic_posts:
34        author = 'anonymous'
35        for user in user_list:
36            if user['id'] == post['author_id']:
37                author = user['name']
38                break
39        print(f'\"{post["title"]}\" by {author}')
```

For each post, the script loops through the list of user records looking for a matching **author_id** value. When it finds a match, the script assigns the associated user name to the **author** variable and breaks out of the loop. The author's name is then printed with the post title.

Serialize the data to reuse it

Suppose you're developing the script and you need to make repeated API requests to test and debug it. This is wasteful when you're dealing with a large data set requiring hundreds if not thousands of requests to get all the data. Instead, you could make just one call, serialize the results, and then reuse the serialized data as many times as you want.

Serializing a data structure means translating it into a format that can be stored and then reconstructed later in the same environment. JSON is a good choice for data returned by the Zendesk API. It also has the added benefit of being human-readable. In Python, you can use the built-in json module to serialize and deserialize a data structure.

Update the script with the highlighted lines to serialize all the post and user data:

```
1   import json
2   import time
3   import requests
4
5   # In production, store credentials in environment variables
6   ZENDESK_API_TOKEN = os.getenv('ZENDESK_API_TOKEN')
7   ZENDESK_USER_EMAIL = os.getenv('ZENDESK_EMAIL')
8   ZENDESK_SUBDOMAIN = 'YOUR_ZENDESK_SUBDOMAIN'
9
10  topic_id = 123456
11  topic_posts = []
12  user_list = []
13
14  url = f"https://{ZENDESK_SUBDOMAIN}.zendesk.com/api/v2/community/topics/{
    topic_id}/posts.json?page[size]=100&include=users"
15
16  auth = f'{ZENDESK_USER_EMAIL}/token', ZENDESK_API_TOKEN
```

```
17
18  while url:
19      response = requests.get(url, auth=auth)
20      if response.status_code == 429:
21          print('Rate limited! Please wait.')
22          time.sleep(int(response.headers['retry-after']))
23          continue
24      if response.status_code != 200:
25          print(f'Error with status code {response.status_code}')
26          exit()
27      data = response.json()
28      topic_posts.extend(data['posts'])
29      user_list.extend(data['users'])
30
31      if data['meta']['has_more']:
32          url = data['links']['next']
33      else:
34          url = None
35
36  topic_data = {'posts': topic_posts, 'users': user_list}
37  with open('my_serialized_data_file.json', mode='w', encoding='utf-8') as f:
38      json.dump(topic_data, f, sort_keys=True, indent=2)
```

The script assigns the user and post data to a new dictionary named **topic**, which it then serializes into a file named **my_serialized_data_file.json** in the current folder.

You can then comment out the rest of the code and deserialize the dictionary as many times as you want to test and format the output:

```
1   import json
2
3   # comment out everything else
4
5   with open('my_serialized_data_file.json', mode='r') as f:
6       topic = json.load(f)
7
8   for post in topic['posts']:
9       author = 'anonymous'
10      for user in topic['users']:
11          if user['id'] == post['author_id']:
12              author = user['name']
13              break
14      print(f'\"{post["title"]}\" by {author}')
```

You can use the same code snippet to develop other scripts without calling the API.

You now have the tools to update your Python scripts to retrieve large data sets with the API. If you want to move your data to Microsoft Excel to view and analyze it, see Paginating through lists using cursor pagination.

Code complete

```python
import json
import time
import requests

# In production, store credentials in environment variables
ZENDESK_API_TOKEN = os.getenv('ZENDESK_API_TOKEN')
ZENDESK_USER_EMAIL = os.getenv('ZENDESK_EMAIL')
ZENDESK_SUBDOMAIN = 'YOUR_ZENDESK_SUBDOMAIN'

topic_id = 123456
topic_posts = []
user_list = []

url = f"https://{ZENDESK_SUBDOMAIN}.zendesk.com/api/v2/community/topics/{topic_id}/posts.json?page[size]=100&include=users"

auth = f'{ZENDESK_USER_EMAIL}/token', ZENDESK_API_TOKEN

while url:
    response = requests.get(url, auth=auth)
    if response.status_code == 429:
        print('Rate limited! Please wait.')
        time.sleep(int(response.headers['retry-after']))
        continue
    if response.status_code != 200:
        print(f'Error with status code {response.status_code}')
        exit()
    data = response.json()
    topic_posts.extend(data['posts'])
    user_list.extend(data['users'])

    if data['meta']['has_more']:
        url = data['links']['next']
    else:
        url = None

topic_data = {'posts': topic_posts, 'users': user_list}
with open('my_serialized_data_file.json', mode='w', encoding='utf-8') as f:
    json.dump(topic_data, f, sort_keys=True, indent=2)

with open('my_serialized_data_file.json', mode='r') as f:
```

```
    topic = json.load(f)

for post in topic['posts']:
    author = 'anonymous'
    for user in topic['users']:
        if user['id'] == post['author_id']:
            author = user['name']
            break
    print(f'\"{post["title"]}\" by {author}')
```