eScholar Ed-Fi API Documentation

Last Updated 2023-10-17

Setting up Ed-Fi API for Data Consumption

Generating bearer token for API calls

- Before being able to pull data from an Ed-Fi ODS via API, you'll need to generate a bearer token for authorization to the API. Below is a code snippet of how we generate this token:

```
def get_access_token():
    secret = get_secret()

    response = requests.post(
        oAuthURL,
        data={"grant_type": "client_credentials"},
        auth=(secret['Key'], secret['SecretKey']),
    )
    response_debug(response)
    return response.json()["access_token"]
```

- Note: generating this bearer token requires you to have an API key and secret key. The code above calls get_secret(), which fetches these credentials from AWS Secrets Manager.

Calling API

- Once the setup for generating an access token is configured, you can now call the API to get data. There are essentially two API calls done per Ed-Fi ODS module. One call retrieves the total count of objects in that module within the min/max change numbers specified. The second call is what actually retrieves that data, iterating in increments of 500 until the total count (retrieved from the first API call) is reached.

Retrieve total count of module

- Below is a code-snippet of how we retrieve the total count of objects in a module. Note, we use 5 different parameters in this call.
    - **offset:** Denotes how many records to skip. We set this to 0.
    - **limit:** Denotes how many records to pull per call. Since this initial call is just retrieving the total count, we set this to 1.
    - **totalCount:** We set this to 'true' as we need the total count field returned.
    - **minChangeVersion:** The lower threshold for pulling deltas.
    - **maxChangeVersion:** The upper threshold for pulling deltas.

```
def get_data(moduleName, schemaColumns, outputBucket, template, MinChangeNumber, MaxChangeNumber):
    s3_client = boto3.client('s3',region_name='us-east-1')
    token = get_access_token()

        #Get total item count for the module
    headers = {
        'accept': 'application/json',
        'authorization': 'Bearer '+token,
    }

    params = {
        'offset': '0',
        'limit': '1',
        'totalCount': 'true',
        'minChangeVersion': MinChangeNumber,
        'maxChangeVersion': MaxChangeNumber,
    }

    response = requests.get(baseRequestURL+moduleName, params=params, headers=headers)
    objects = response.headers["total-count"]
```

Pulling Data

- Now, the total count of the module is stored in the **objects** variable. We can now iteratively call the API to get data from a specific module:

```
offset = 0
limit = 500
    #iteratively call the API until you've queried all available items
while offset <= int(objects):
    #Getting new bearer token every 50k records
    if offset % 50000 == 0 and offset != 0:
        token = get_access_token()

    headers = {
        'accept': 'application/json',
        'authorization': 'Bearer '+token,
    }

    params = {
        'offset': str(offset),
        'limit': str(limit),
        'totalCount': 'false',
        'minChangeVersion': MinChangeNumber,
        'maxChangeVersion': MaxChangeNumber,
    }

    ## actual API call
    r = requests.get(baseRequestURL+moduleName, params=params, headers=headers)

    #Logs for data pull
    json_response(r,offset,moduleName)

    data = r.json()

    dic_flattened = [flatten(d) for d in data]
    df1 = pd.DataFrame(dic_flattened, columns=schemaColumns)

    ## places the results of the API call into a bucket
    if len(df1.index) >= 1:
        s3_client.put_object(Body=df1.to_csv(index=False), Bucket=outputBucket, Key= template + '/' + moduleName+'/'+moduleName+str(offset)+'.csv')

    ## updates the offset by the defined limit to ensure you query all the available data
    offset = offset + limit
```

- Note: We use the same 5 parameters in this call, but the values of the offset and limit are variable based on the iteration of the API call. Every call, the offset gets incremented by 500 until the offset is greater than the total count (objects variable)
- Being that the bearer token can timeout, we generate a new bearer token every 50,000 records.

Dealing with JSON formatting

- Being that the requests are returned in an unstructured JSON format, we need to be able to flatten this data and only return the fields we need to ingest into our data warehouse. To accomplish this, we use a schema CSV that denotes all necessary fields needed from a module.
- These lines from the snippet above flatten the data from the response and makes sure that only necessary fields are included in the output when we convert the output to a pandas DataFrame

```
dic_flattened = [flatten(d) for d in data]
df1 = pd.DataFrame(dic_flattened, columns=schemaColumns)
```

- Note: We utilize the flatten-json python library to take care of the flattening of the JSON response
- The data is now ready to be transformed and mapped to our pre-defined templates for ingesting into our CDW.