Third Party API Calls with Extension Framework

This walk through assumes that you are using a third party api that supports api key and secret based authentication (the examples leverage the petfinder api, if you'd like to follow along with the examples I'd recommend generating api credentials here -> https://www.petfinder.com/developers/)

Once the api key and secret are generated, we will follow these next steps:

1. Define Scoped user attributes (*and key names to use for these attributes*)
2. Create an extension namespaced user attribute representing api key and secret and assign to either your user or a group
3. Use ***createSecretKeyTag()*** method to safely pull user attributes from Looker server
4. Use ***serverProxy()*** method to resolve api keys in the Looker server and return an access token
5. Use ***fetchProxy()*** method to pass access_token and make calls to third party api endpoints

---

Scoped User Attributes

first define a scoped user attribute param in your extension entitlements (https://docs.looker.com/data-modeling/extension-framework/js-r-extension-examples#user_attributes).

Decide on what the key names will be for both the api key (*client id*) and client secret. (*In the example here client id is "pet_client_id" and the secret is "pet_client_secret*). Once these are defined, add them to the scoped user attributes param in your extension's entitlements.

Example:

```
application: ef_education {
  label: "Luka Demo: Admin Utility"
  url: "http://localhost:8080/bundle.js"
  # file: "bundle.js"
  entitlements: {
    use_embeds:  yes
    use_form_submit: yes
    new_window: yes
    core_api_methods: ["me","update_scheduled_plan","run_query","all_scheduled_plans"]
    external_api_urls: ["https://api.petfinder.com/*","https://api.petfinder.com/v2/animals"]
    scoped_user_attributes: ["pet_client_id","pet_client_secret"]
  }
```

}

Defining Extension Namespaced User Attributes

After defining the scoped user attributes, navigate to the Admin -> User -> User Attributes page (*you will need to be an admin on the instance to perform these next steps*). Here we are going to be creating two user attributes one for the api key (*client id*) and one for the client secret. Because these are scoped user attributes, both of these will need to be namespaced with the key name's defined earlier to the extension.

For example, in the example extension this is the full url (https://<hostname>/extensions/luka_thesis_extension::ef_education). The extension "id" is the "luka_thesis_extension::ef_education" portion of the url. A namespaced user attribute takes the extension id, replaces the "::" with a single underscore "_" and then appends the key name. Following the example thus far we'd define two user attributes named:

- luka_thesis_extension_ef_education_pet_client_id
- luka_thesis_extension_ef_education_pet_client_secret

For both of these user attributes they will have the following settings:

- Data Type: String
- User Access: None
- Hide Values: Yes
- Domain Allowlist: api url of the client, in this example that would be https://api.petfinder.com/*

Then we will assign a default value to these attributes or a specific value to a given user or group of users.

Extension SDK

This will be all we need to do in the Looker UI to get setup. Next will be to move to our extension source code. We will be making use of three different extensionSDK methods:

- **createSecretKeyTag()**: takes in a scoped user attribute name, substitutes that for the namespaced user attribute {{extension_id_key_name}} and the Looker server resolves this to the actual user attribute value, this is not surfaced in the browser itself.
- **serverProxy():** serverProxy is a method that accesses the endpoint via the Looker server. This mechanism allows client IDs and secret keys to be set securely by the Looker server. This method is used for authentication and is used to return an access token to be used later on.

- **fetchProxy():** fetchProxy is a method that accesses the endpoint via the user's browser. It will take in the access token fetched from the serverProxy to make consecutive api calls.

---

createSecretKeyTag

Starting with the createSecretKeyTag() method, we are going to make two calls; one to the client_id and one to client_secret. Following our example that would look like this:

```
const client_id = extensionSDK.createSecretKeyTag('pet_client_id')
const client_secret = extensionSDK.createSecretKeyTag('pet_client_secret')
```

If we print out the variables client_id and client_secret, respectively they would return {{luka_thesis_extension_ef_education_client_id}} & {{luka_thesis_extension_ef_education_client_secret}} both of which would be resolved by the looker server to their values.

---

serverProxy

Next we are going to make a call to the server proxy method. Before doing so we are going to want to first make sure we have the api base url with a wildcard, allowlisted in the external_api_urls parameter of our extension. For example:

external_api_urls: ["https://api.petfinder.com/*"]

The server proxy call typically will follow this format:

```
serverProxy(api auth url,
{
 method,
 headers: {}, // typically this will include the content-type header
 body
})
```

For the petfinder example we are following, the call to the server proxy method might look like this:

```
const petFinderAuth = await extensionSDK.serverProxy(
    'https://api.petfinder.com/v2/oauth2/token',
    {
```

```
      method: 'POST',
      headers: {
        'Content-Type': 'application/x-www-form-urlencoded'
      },
      body: 'grant_type=client_credentials&client_id=' + client_id +
'&client_secret=' + client_secret
    })
    const {access_token} = petFinderAuth.body
```

---

fetchProxy

Assuming successful authentication, the body of the response should return an access token, which we can use in the fetchProxy() method.

Depending on what http request method is used the format for the fetchProxy will change, however the general fetchProxy call format is as follows:

```
fetchProxy(api endpoint url,
{
  method,
  headers: {content_type, authorization},
  body // this is optional and will depend on the http request method used
}
```

Following our example a get request to one of petFinder's api endpoints might look like this:

```
const getPets = await extensionSDK.fetchProxy(
        'https://api.petfinder.com/v2/animals?type=dog&page=2',
        {
          method: 'GET',
          headers: {
            'Content-Type': 'application/JSON',
            Authorization: `Bearer ${access_token}`
          }
        }
      )
      console.log(getPets)
```