Les Chroniques du Campagnol
Season 2 Episode 8
Anonymous access to LabX

May 7, 2022

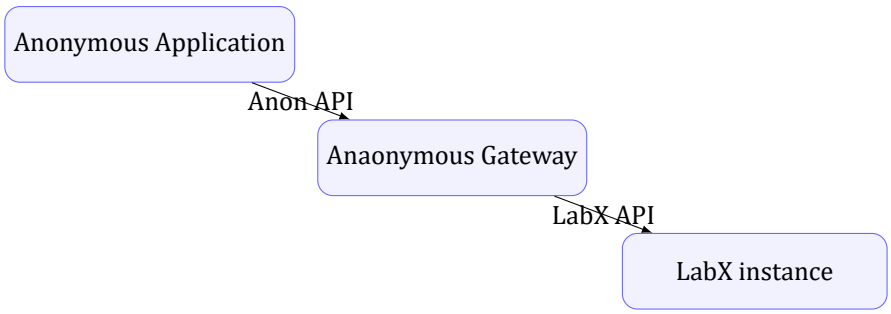
One of the biggest problems of the LabX environment is that user management is not quite fine grained, and it's mostly manual also.

This makes it difficult to provide access to metric upload from untrusted applications. The current system either allows unrestricted access to the API, or no access at all.

1 Anonymous application

I define an **Anonymous Application** as an application that accesses the LabX API through a simplified portal that does not require identification through Auth0, and uses long-lived API keys instead. That anonymous application does not have any implicit right on the data, and can only access a very limited subset of the endpoints. That subset needs to be explicitly made accessible to the application.

I define an **Anonymous Gateway** as an application that channels the anonymous API calls to genuine LabX API calls, applying additional authorization rules. The Anonymous Gateway must know a LabX username and password to be allowed access to the data, but those credentials remain secret.



2 User needs

What I want is a way to give access to some metrics and some KPIs to an anonymous application. The anonymous application would be able to upload the values of those metrics, and read the values of the KPIs, and that's all. Especially, what the anonymous application cannot do is:

- Change the setup in any way.
- Query the setup in any way.
- Upload anything else than the metrics for which it has explicitly received an upload grant.
- Download anything else than the KPIs for which it has explicitly received a download grant.

What I want also is a simplified way to allow an anonymous application to access the API, without requiring the administrator to go through the whole process of registering a new user into Auth0. Doing this will also greatly simplify authentication, as it will not require the API to use an Auth0 flow. The protocol will be more like an M2M flow.

Note: We do not use Auth0, but that does not mean that *any* anonymous application can arbitrarily use the API. An administrator must first setup access rights and access tokens inside the LabX database before the gateway will allow the application to access data.

3 Anonymous API

The API will initially provide two, and only two endpoints.

3.1 POST /api/anonymous/1.0/metrics

This enpoint allows an anonymous application to upload the value of resource metrics. The payload includes:

- Authentication information.
- A collection of metric values, each comprised of:
  - The identity of the resource, in the form of its internal GUID.
  - The identity of the metric kind, in the form of its internal GUID.
  - The value for the metric.
  - The (optional) validity time.

The application cannot force the identity of the metric uploader (it is always the authenticated user).

The gateway is free to rate-limit the metric uploads. Especially, it is allowed to extend the validity time if the application has requested a value too low, temporarily cache values before transmitting them to LabX, and to ignore uploads of values that are too close to the previous one.

The gateway is also free to isolate the anonymous application from transient LabX errors such as errors 502 or 503.

3.2 POST /api/anonymous/1.0/kpis

This endpoint allows an anonymous application to retrieve the current values of KPIs. The payload includes:

- Authentication information.
- A collection of KPI ids for which the value is requested.

The application can only query KPIs that it is explicitly allowed to read, and it cannot use active wait nor force value recomputation; the application must poll for values and obey proposed refresh rates; it cannot subscribe to websocket notifications.

Note: While the endpoint is used to read data, it still uses the `POST` HTTP verb because we need to transmit authentication information in the body of the call.

The gateway is free to rate-limit the KPI queries. Especially, unusually long lists of queries shall be truncated, re-querying a KPI value that has not reached its refresh time is likely to return a cached value instead of attempting a recomputation.

3.3 Authentication

Authentication is part of the JSON payload and is not provided in HTTP headers. The full authentication credentials must be provided in each request; there is no initial step of calling an `/auth` endpoint to obtain a JWT token.

Authentication consists of:

- The Tenant ID.
- The User ID in the form of the GUID of a resource. That resource must exist in the tenant database, but does not need to be linked to an Auth0 account. It does not need to be of kind "User" (it can be a connection, a process, a workstation, etc.). When uploading metrics, the user ID will serve as the metric provider.
- The API Keys, in the form of arbitrary secret strings associated with the user[1] and with the tenant.

3.4 Obtaining the GUIDs

The application must already know the resource and metric kind identities. It cannot query the database in any way to retrieve them. Typically, a user would log-in into LabGPS or any other LabX online application to retrieve those using the standard API calls to RESOURCE, METRIC or KPI microservices, and somehow copy-and-paste them into the application.

3.5 Why not a little example payload?

This is the kind of stuff I intend to pass to the metrics endpoint:

```
1   {
2       "tenant": "TENANT4a0cba230a5e405980f10af48fc8c2ac",
3       "tenantkey": "f563c5c4eba9464fb753019e50ca980b90a74045",
4       "user": "7c8d6bf6-76ba-4998-9890-6833b4d80ee6",
5       "userkey": "47821e3fad9766ae4c3447bf8927794046132cd5",
6       "79c5633d-8214-438a-9253-2e2c12d91d8a":
7       {
8           "f63c70f4-edc6-44ed-9dc4-cd38bfb2dca8": 20,
9           "07d250d2-5e79-4b59-8b26-1f58fae37f11": 30
10      },
11      "3acaff03-41d2-4045-9c14-096459e7605e":
12      {
13          "0b662908-5eb8-4493-8c27-67b826b485a7":
14          {
15              "value": 20,
16              "validity": 3600
17          },
18          "bc936336-6e0a-4c3b-9f7e-51460545e0db": [30, 3600]
19      }
20  }
```

As you can see, I intend the structure to be as shallow as possible. The root element contains the four components of the authentication credentials (lines 2 to 5), and a list of resource IDs for which we want to upload metrics (lines 6 and 11). Inside each resource is a list of metrics to upload, in the form of the metric kind id and the value. Lines 8 and 9 show the abbreviated way, where you just give the value itself.

Line 12 shows an example of a value and its validity, expressed explicitly. Line 17 shows a shorthand notation where the value and validity are passed as an array instead of a map.

3.6 Future additional endpoints

It may prove useful, in the future, to provide a few additional endpoints. For example, an anonymous application could upload resource properties, audit log entries, or query for pending operations.

4 Authorization

As I said earlier, the gateway itself is trusted (it knows a Labx username and password) but the anonymous application is not. The gateway uses the credentials from the anonymous API, and uses them to retrieve authorization data stored in LabX as *Resources* and *Resource Links*:

[1] Well... yeah, you can call that a *password* if you like.

- The untrusted application is represented by a resource. The authentication key is a property of that resource.
- The resource is a master (with Resource Link Kind Read Access) of all the KPIs the application can query.
- The resource is a master (with Resource Link Kind Write Access) of all the other resource for which it can upload metrics.
- The resource has a custom property listing all the Metric Kinds it is allowed to upload.

This will, once again, put the resource architecture under scrutiny, and prove that it can handle such a scenario. The gateway shall of course use local storage to cache the information for some time, so that each anonymous API call does not translate in the whole resource structure to be re-queried each time.

5 But... Why?

There are two main reasons to provide this anonymous API:

- First, because this is a solution to some actual problems. We truly want some applications to upload data or display KPIs in a simplified manner. And more importantly, we do not necessarily *trust* those applications with actual Auth0 credentials, for example if they are generic applications running on a phone. Although I have never heard of one, I'm pretty sure that there are apps on Android and iPhone that are able to query simple endpoints for data.
- Second, and more importantly, because it will allow me to learn a boatload of new features...
  - **App Runner**, a simple, managed, serverless application hosting framework from AWS. This should be an order of magnitude easier to deploy than our current heavy-duty LabX infrastructure.
  - **Copilot**, **Code Pipeline** and other AWS CI/CD tools.
  - Writing Web Applications in C♯.
  - Running C♯ and .NET under Linux.
  - Containerizing C♯ applications.

What I want to do is develop the gateway as a standalone containerized application written in C♯.

6 Deploying the application

As a starting point, I will use that article: Deploy .NET 6 API to AWS App Runner using AWS Copilot CLI.

And just in case you wonder, the gateway application is likely to be called ffawkes. One day I might even tell you why - and as usually there is more than one reason, I suppose that you got the obvious reference but I also guess you missed the ironic ones. - Keep reading my chronicles for more information.

7 What's next

Next, I will start to research ways to create anonymous clients for multiple platforms. Obvious paths are data collectors à la Grafana, but I'll try to also create some desktop clients, maybe a Visual Studio Code extension, some desktop applets, etc. Something nice to have would be a dashboard application like the one we had in LPM, which docks on an edge of the screen and displays some statistics.