OAuth2 Authentication with Lua (http://lua.space/webdev/oauth2-authentication-with-lua)

By Israel Sotomayor Jan 14 2016 14:50 Webdev (/webdev) Reblogged (https://moltin.com/blog/2016/01/oauth2-authentication-with-lua) 1 (http://lua.space/webdev/oauth2-authentication-with-lua#disqus_thread)

Just to clarify, this won't be a detailed technical guide about how you can build your own authentication layer using OpenResty (https://openresty.org) + Lua (http://www.lua.org), rather it is a document explaining the process behind the solution.

This is a real example of how moltin (https://moltin.com)'s API has come to rely on OpenResty (https://openresty.org) + Lua (http://www.lua.org) to handle our oauth2 authentication for all users.

The logic used to authenticate a user was originally embedded into moltin (https://moltin.com)'s API, built using the PHP Framework Laravel (https://laravel.com/). This means a lot of code had to be booted before we could authenticate, reject or validate a user request resulting in a high latency.

I'm not going to give details about how much time a PHP Framework could take to give a basic response, but if we compare it to other languages/frameworks, you can probably understand.

This is roughly how it looked at that time:

```
...
public function filter($route, $request) {
    try {
        // Initiate the Request handler
        $this->request = new OAuthRequest;
        // Initiate the auth server with the models
        $this->server  = new OAuthResource(new OAuthSession);
        // Is it a valid token?
        if ($this->accessTokenValid() == false) {
            throw new InvalidAccessTokenException('Unable to validate ac
cess token');
        }
...
```

So we decided to move all logic one layer up to OpenResty (https://openresty.org) + Lua (http://www.lua.org) which achieved the following:

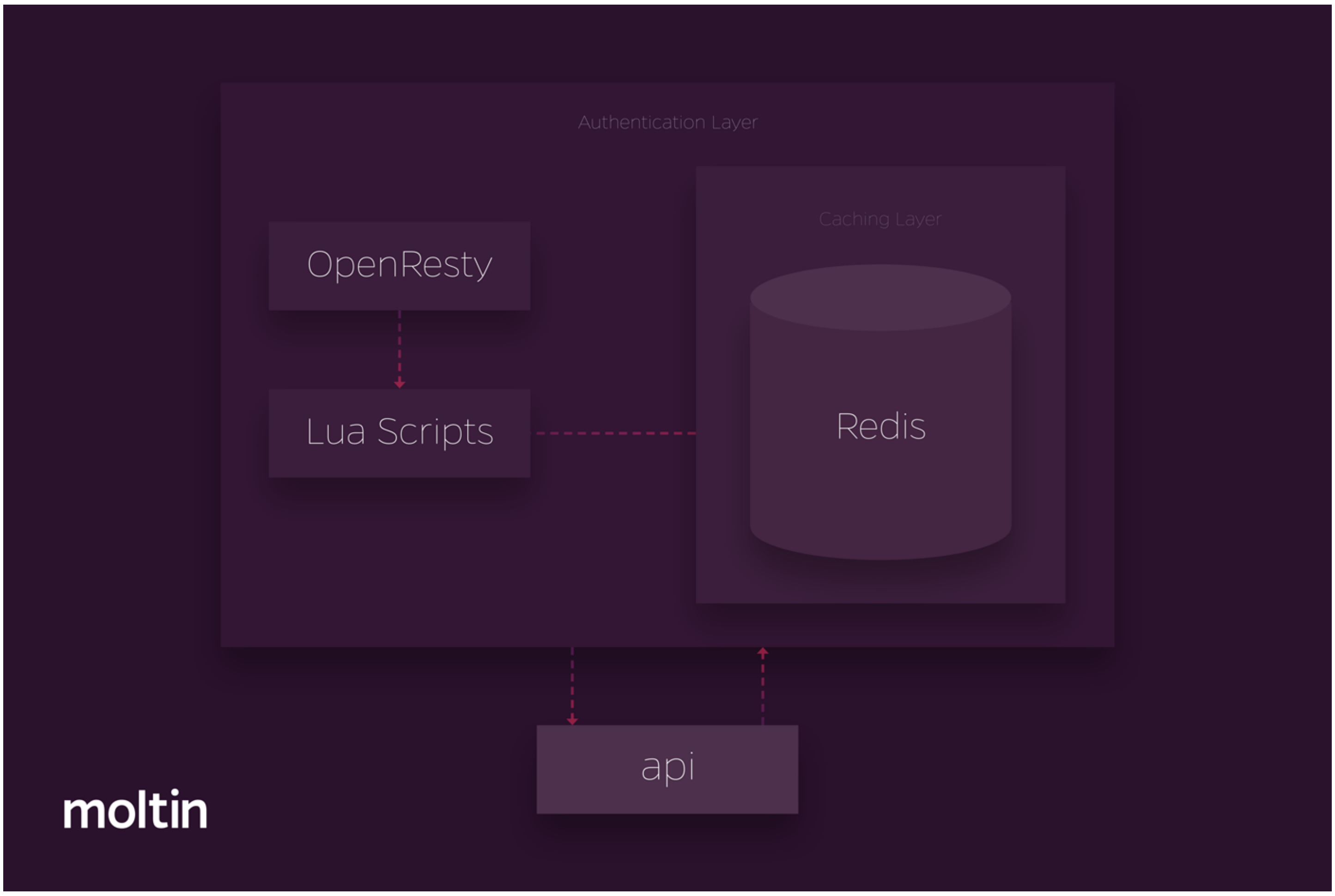
- Decoupled responsibilities from our Monolitic API.
- Improved authentication times and generated access/refresh tokens.
- Improved times for rejecting invalid access tokens or authentication credentials.
- Improved times when validating an access token and redirecting the request to the API.

We wanted, and needed, to have more control on each request before hitting the actual API and so we decided to use something fast enough to allow us to pre-process each request and flexible enough to be integrated into our actual system. Doing this led us to use `OpenResty` , a modified version of Nginx (https://www.nginx.com/), that allowed us to use Lua (http://www.lua.org), the language used to pre-process those requests. Why? Because it's robust and fast enough to use for these purposes and it's a highly recognised scripting language used daily by many large companies.

We followed the concept behind Kong (https://github.com/Mashape/kong), who use OpenResty (https://openresty.org) + Lua (http://www.lua.org), to offer several micro-services that could be plugged into your API project. However we found that Kong is in a very early stage and is actually trying to offer more than what we needed therefore we decided to implement our own Auth layer to allow us to have more control over it.

Infrastructure

Below is how moltin (https://moltin.com)'s infrastructure currently looks:



- OpenResty (Nginx)
- Lua scripts

- Caching Layer (Redis)

OpenResty

This is the bit that rules them all.

We have routing in place to process each of the different user's requests as you can see below:

nginx.conf

```
location ~/oauth/access_token {
    ...
}
location /v1 {
    ...
}
```

So for each of those endpoints we have to:

- check the authentication access token
- get the authentication access token

```
...
location ~/oauth/access_token {
    content_by_lua_file "/opt/openresty/nginx/conf/oauth/get_oauth_access.lua";
    ...
}

location /v1 {
    access_by_lua_file "/opt/openresty/nginx/conf/oauth/check_oauth_access.lua";
   ...
}
...
```

We make use of the OpenResty directives content_by_lua_file (https://github.com/openresty/lua-nginx-module#content_by_lua_file) and access_by_lua_file (https://github.com/openresty/lua-nginx-module#access_by_lua_file).

Lua Scripts

This is where all the magic happens. We have two scripts to do this:

get_oauth_access.lua

```lua
...
ngx.req.read_body()
args, err = ngx.req.get_post_args()

-- If we don't get any post data fail with a bad request
if not args then
    return api:respondBadRequest()
end

-- Check the grant type and pass off to the correct function
-- Or fail with a bad request
for key, val in pairs(args) do
    if key == "grant_type" then
        if val == "client_credentials" then
            ClientCredentials.new(args)
        elseif val == "password" then
            Password.new(args)
        elseif val == "implicit" then
            Implicit.new(args)
        elseif val == "refresh_token" then
            RefreshToken.new(args)
        else
            return api:respondForbidden()
        end
    end
end

return api:respondOk()
...
```

check_oauth_access.lua

```lua
...
local authorization, err = ngx.req.get_headers()["authorization"]

-- If we have no access token forbid the beasts
if not authorization then
    return api:respondUnauthorized()
end

-- Check for the access token
local result = oauth2.getStoredAccessToken(token)

if result == false then
    return api:respondUnauthorized()
end
...
```

Caching Layer

This is where the created access tokens are stored. We can remove, expire or refresh them as we please. We use Redis as a storage layer and we use openresty/lua-resty-redis (https://github.com/openresty/lua-resty-redis) to connect Lua to Redis (http://redis.io/).

Resources

Here are some interesting resources on Lua that we used when creating our authentication layer.

Lua

- Lua formdata type (http://blog.zot24.com/lua-formdata-type/)
- Lua sugar syntax double dots (http://blog.zot24.com/lua-sugar-syntax-double-dots/)
- How to use a classs constructor on Lua (http://blog.zot24.com/how-to-use-a-classs-constructor-on-lua/)
- Returning status code with OpenResty Lua (http://blog.zot24.com/returning-status-code-with-openresty-lua/)
- Return JSON responses when using OpenResty + Lua (http://blog.zot24.com/return-json-responses-when-using-openresty-lua/)
- When ngx exit using OpenResty precede it with return (http://blog.zot24.com/when-ngx-exit-using-openresty-precede-it-with-return/)

About Israel Sotomayor

(zot24.com) (http://twitter.com/zot24) (http://github.com/zot24)

Software developer at Moltin