REST API: Authorization with curl

REST API · https://mriacrm.com/docs/rest-api-authorization-with-curl/

The whole OAuth 2.0 (3LO) handshake as plain HTTP requests, run against a test site: the authorize URL, the callback, the code exchange, the first API call, the refresh, and what each step answers when something is wrong. REST API: Setup creates the integration these requests use; REST API: Example Postman and REST API: Example Node.js run the same handshake for you. This page is for when you write the client yourself, or need to see why a token does not work.

1. What you need

Value	Where it comes from
`client_id`, `client_secret`	The integration's **Settings** tab in the developer console (REST API: Setup).
Callback URL	Registered on the integration's **Authorization** tab, character for character. This page uses `http://localhost:8765/callback`.
`cloudId`, `envId`	Both are in the Base URL that admin.atlassian.com shows under App REST APIs: `https://api.atlassian.com/svc/jira/<cloudId>/apps/f3bb4fc8-7911-405b-959b-002b3292a3d0_<envId>/v1`. The first UUID is the cloudId; the UUID after the underscore is the environment id of the Mria CRM install on that site.
A user with a Mria CRM role	The handshake runs in that user's browser. Every call the token makes afterwards carries that user's CRM permissions.

2. The authorize URL

Send the user's browser to this URL. Line breaks are for reading; the parameters are URL-encoded on one line.

```
https://auth.atlassian.com/authorize?
  audience=api.atlassian.com
```

```
  &client_id=<client_id>
  &scope=read:crm:custom%20write:crm:custom%20read:profile:custom%20read:forge-
app:jira%20offline_access
  &redirect_uri=http://localhost:8765/callback
  &state=<random string you keep>
  &response_type=code
  &prompt=consent
  &sns=f3bb4fc8-7911-405b-959b-002b3292a3d0.<envId>
```

Parameter	Value	What it does
`audience`	`api.atlassian.com`	Names the API gateway the token is for. The OpenAPI specification and the Node.js example both send it; leaving it out does not change the consent screen, so a missing value is not caught at this step.
`scope`	The Mria CRM scopes, `read:forge-app:jira` and `offline_access`	`read:forge-app:jira` must be in the list: without it Atlassian replaces the consent screen with *Something went wrong* and tells the app owner “This app has not requested any supported Atlassian scopes”. `offline_access` is what returns a refresh token. A scope the integration does not have is dropped silently.
`redirect_uri`	A registered callback URL	Must match a callback on the integration exactly. A URL that is not registered stops the flow before any consent screen (section 7).
`state`	A random string	Comes back unchanged on the callback. Compare it before you exchange the code.
`response_type`	`code`	Authorization-code flow. Nothing else is supported.
`prompt`	`consent`	Shows the consent screen. Atlassian's 3LO guide marks it required.

Parameter	Value	What it does
`sns`	`<appId>.<envId>`	Mria CRM's app id and the environment id of the install, joined with a dot (the API path joins the same two ids with an underscore). This is what puts the Mria CRM scopes on the consent screen: without `sns`, or with an environment id that is not installed on the site, the screen only asks for “View Forge app” and the token cannot reach the CRM.

What the user sees

The consent screen names the integration, asks the user to pick a site (“Install app on”) when the integration is resource-level, and lists the grants in two groups: *In Jira: View → Forge app* and *In Mria: Access → Read CRM data, Read profile, Manage CRM data*. An integration that is still in development mode carries a warning box. The user clicks **Accept**.

The callback

The browser is redirected to the callback URL with the code and your state:

```
GET http://localhost:8765/callback?state=<your state>&code=<authorization code>
```

The code is single-use; exchange it straight away.

3. Exchange the code for tokens

From your backend — this is the only request that carries the client secret:

```
curl -s -X POST https://auth.atlassian.com/oauth/token \
  -H 'Content-Type: application/x-www-form-urlencoded' \
  -d grant_type=authorization_code \
  -d client_id=$CLIENT_ID -d client_secret=$CLIENT_SECRET \
  -d code=$CODE \
  -d redirect_uri=http://localhost:8765/callback
```

The response (tokens shortened; both are JWTs of two to three thousand characters):

```
{
 "access_token": "eyJraW…mi1Q (3081 chars)",
 "expires_in": 3600,
 "refresh_token": "eyJraW…_dlg (2400 chars)",
 "scope": "<appId>.<envId>:read:crm:custom <appId>.<envId>:read:profile:custom
<appId>.<envId>:write:crm:custom offline_access read:forge-app:jira",
 "token_type": "Bearer"
}
```

`expires_in` is in seconds: the access token lives one hour. `scope` lists what was granted, and the Mria CRM scopes come back prefixed with the app and environment they are bound to — that prefix is why a token works on one install only (section 7). The refresh token is only present when `offline_access` was requested. Store both tokens server-side; the refresh token is as sensitive as the secret.

4. The first call

`GET /me` needs no CRM permission, so it is the call to verify a token with. It returns the user's account id and effective CRM permissions (`"*"` for an administrator; a list of permission keys otherwise):

```
curl -s "https://api.atlassian.com/svc/jira/<cloudId>/apps/f3bb4fc8-7911-405b-959b-
002b3292a3d0_<envId>/v1/me" \
  -H "Authorization: Bearer $ACCESS_TOKEN"
```

```
{
 "accountId": "712020:<account-id>",
 "permissions": [
  "*"
 ]
}
```

5. Refresh

Refresh before the hour is up, or on the first 401:

```
curl -s -X POST https://auth.atlassian.com/oauth/token \
  -H 'Content-Type: application/x-www-form-urlencoded' \
  -d grant_type=refresh_token \
```

```
  -d client_id=$CLIENT_ID -d client_secret=$CLIENT_SECRET \
  -d refresh_token=$REFRESH_TOKEN
```

The answer has the same shape as the exchange: a new access token, a new `expires_in` of 3600, and a *new* refresh token that differs from the one you sent.

Refresh tokens rotate. Every refresh returns a new refresh token. Atlassian's refresh-token guide says the one you used is disabled; in our test a second use a few seconds later was still answered with a fresh pair, so do not read a 200 as proof that you kept the right token. Always store the newest refresh token before you use the new access token — a client that keeps an old one is locked out at a later refresh. A token that is not a refresh token at all answers:

```
HTTP 403
{
 "error": "unauthorized_client",
 "error_description": "refresh_token is invalid"
}
```

6. Putting it together

A minimal client, as a sequence:

1. Build the authorize URL of section 2 with a fresh `state` and open it for the user.
2. On the callback, compare `state`, then POST the code to `/oauth/token` (section 3). Save `access_token`, `refresh_token` and the expiry time.
3. Call `GET /me` once (section 4) and keep the `accountId` — it is the user every later write is attributed to.
4. Before each call, refresh if the access token is within a minute of expiry (section 5); after a 401, refresh once and retry once.

The Node.js example is exactly this in one file; the Daily Lead Sync page has the same four steps in Python, with the tokens kept in a file.

7. When it goes wrong

Each row was reproduced against a test site on 2026-11-06. The first group stops before the consent screen; the second comes from the token endpoint; the third shows at the API.

You did	You get
`redirect_uri` not registered on the integration	Redirect to `id.atlassian.com/error` with `error=unauthorized_client` and `error_description=redirect_uri is not registered for client: <the uri you sent>`. The page reads “The app's callback URL is invalid.”
Wrong `client_id`	`error=invalid_request`, `error_description=failed to retrieve client` — “We couldn't identify the app requesting access.”
No `read:forge-app:jira` in `scope`	“Something went wrong” instead of the consent screen. Under *Information for the owner*: “This app has not requested any supported Atlassian scopes. Check the authorization URL for your app and ensure that it includes valid scopes.”
No `sns`, or an environment id that is not installed on the site	The consent screen lists only *In Jira: View Forge app*. The Mria section is missing, so the CRM scopes are never granted and the token cannot pass the gateway's scope check for the CRM paths (the 403 below). Observed on the consent screen; that token was not exchanged.

The same code exchanged twice	`HTTP 400`
	`{`
	`"error": "invalid_grant",`
	`"error_description": "authorization_code is invalid"`
	`}`

`redirect_uri` on the token request differs from the authorize request	`HTTP 400`
	`{`
	`"error": "invalid_client",`
	`"error_description": "redirect_uri is not registered for client: http://localhost:8765/other"`
	`}`

You did	You get

No `Authorization` header, or a token that is expired, revoked or not a token	From the gateway, not from Mria CRM:
	`HTTP 401`
	`{`
	`"code": 401,`
	`"message": "Unauthorized"`
	`}`

A token consented with one environment's `sns` used on another environment's path	Also from the gateway; a plain string, not JSON:
	`HTTP 403`
	`"Missing required scopes for path=/v1/me, method=GET. Missing: [read:crm:custom]"`
	The `envId` in the path, the one in `sns` and the environment the integration was granted on must be the same install.

App REST APIs not enabled for the site	`403 REST APIs are not allowed for appId=…` — the admin step in REST API: Setup was skipped. Read from the specification, not reproduced: switching the API off would have interrupted the other clients on the test site.

Related Pages

- REST API: Setup — enabling the API and creating the integration the handshake uses.
- REST API: Paging, Filters, Errors and Limits — what the API answers once you hold a token.
- REST API: Example Node.js — the handshake in code.
- Implementing OAuth 2.0 (3LO) and Implementing the Refresh Token Flow — Atlassian's own description of the flow.

This API is supported and actively developed, so it can change without notice. It builds on Atlassian's Forge App REST APIs (Preview). Send feedback to the Mria CRM team.