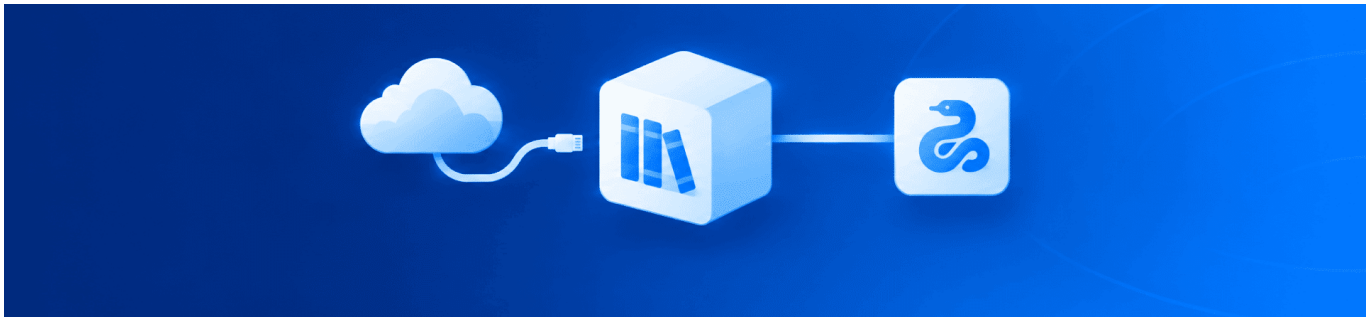
Calling a PHP library from Python: an HTTP SDK or an embedded bridge

LibreDTE Lib Core can be reached from Python in two ways, both built on Derafu Backbone: an HTTP SDK, and a bridge that runs PHP inside the Python process. What each needs, what each costs, and the license question that decides it for many projects.



Derafu · 17/11/2026 · #backbone, #python, #architecture

Calling a PHP library from Python: an HTTP SDK or an embedded bridge

Derafu DEV builds Derafu Backbone, and a generic Python bridge for it. LibreDTE Lib Core, a free library for electronic invoicing in Chile, is built on Backbone and uses both to be reachable from Python. This post is about the two ways in that came out of that, how they differ and how to choose.

The problem: a PHP library, a Python application

LibreDTE Lib Core is written in PHP. A Python application that needs it, a Django one for example, has two ways in. Rewriting a library of 51 operations is not one of them.

Both ways start from the same place. In Backbone, everything that a library lets you do is an **operation** with an id, such as `billing.identifier.caf_loader::load`. It loads a CAF, the file that Chile's tax authority gives an issuer to authorize a range of invoice numbers, and it gives back the range and the validity already resolved. An operation takes plain parameters and answers with a result that any caller can read. What changes between the two ways is only how the Python side gets to it.

Way one: over HTTP

Lib Core is published as an HTTP API, in a container image, that Derafu Backbone API builds from those same operations. The path says which operation to call, and the parameters go in the body:

```
POST /api/billing/identifier/caf_loader/load
Content-Type: application/json

{ "parameters": { "xml": "<the CAF, in base64>" } }
```

You can make that call with any HTTP client, but there is also an SDK, `libredte-lib-sdk`, that is MIT licensed, uses `httpx` and mirrors the layout of the API. It gives you typed results instead of dictionaries:

```
from libredte_lib_sdk import LibreDTE

# The base URL and the token come from arguments or from environment variables,
# so the SDK can point to a container of your own.
sdk = LibreDTE()

caf = sdk.billing.identifier.caf_loader.load(xml_base64)

caf.folio_desde    # A typed object, not a dictionary.
caf.folio_hasta
caf.vigente
```

When a call fails, the SDK raises an exception that carries the HTTP status, and it has its own ones for a lost connection and for a rate limit.

Way two: PHP inside the Python process

The other way has no network. The bridge, `libredte-lib-core-bridge`, loads the PHP engine **inside your own Python process**, with the `phpy` extension of Swoole, and calls the dispatcher of the library directly. It is built on `derafu-backbone-bridge`, the generic Python bridge of Backbone, which is MIT licensed and knows nothing about any specific library.

```
from libredte_lib_core_bridge import Dispatcher

dispatcher = Dispatcher()

result = dispatcher.dispatch(
    'billing.identifier.caf_loader::load',
    xml=xml_base64,
)
```

```
result.value      # A plain Python dictionary.
result.metadata   # Time, memory and process of the call.
```

A call never ends in a raw PHP error. It raises a Python exception, of a class that extends `BackboneBridgeError`, and it carries the real problem and the metadata of the call.

The price is the environment. You need Python 3.14 or later, built with `phpy`, and `phpy` needs a PHP built with `--enable-embed` that is installed in the system: it is not installed with pip. There is a reference image with all of that ready, and the dispatcher of the library has to be installed with Composer and found through an environment variable or an argument. The bridge is used already by a Python web application.

Two ways into the same operations



	HTTP SDK	Embedded bridge
Where PHP runs	In a separate service, a container.	Inside your Python process.
What you install	The SDK, and an API to call, yours or someone else's.	Python 3.14 built with `phpy`, a PHP built with `--enable-embed`, and the dispatcher with Composer.
Network	One HTTP call for each operation.	None.
The result	Typed objects.	Plain Python data, and the metadata of the call.
The errors	An exception with the HTTP status.	An exception with the problem and the metadata.
License and your application	It does not spread. Your application only calls a service, so the license of the library does not reach it. The SDK is MIT.	It spreads. The library runs inside your process, so the AGPL applies to your application as if you used the PHP library directly. The bridge is AGPL-3.0.
Platforms	Anywhere that runs Python and HTTP.	Linux and macOS.

The decision that is not technical

There is one row of that table, the one about the license, that decides it for many projects, and it is not about technology. Lib Core, its API, its dispatcher and its bridge are licensed under the AGPL-3.0. The documentation of the bridge says it plainly: since PHP runs inside your Python process, your application is as subject to the AGPL as if you used the PHP library directly. For those who want to integrate without that effect, the same documentation points to the HTTP API and to the Docker images.

That is what that documentation says. It is not legal advice, and you should read the license of every piece that you embed or call. The generic bridge of Backbone is MIT: the AGPL comes from the library that you load into the process.

How to choose

- **Choose HTTP** when the license matters to you, when you cannot install a PHP with `--enable-embed` where the application runs, when several applications or languages share one service, or when you want typed results out of the box.
- **Choose the bridge** when the terms of the AGPL are fine for your project, and you want no service to run and no network between the application and the library.

What it costs

- **The bridge is at an early version.** It is at 0.2, and the LibreDTE bridge depends on `derafu-backbone-bridge~=0.2.0`. `phpy` needs a system-wide PHP build, so your Docker image or your server has to be prepared for it.
- **The HTTP status codes are fixed for now.** The API maps a missing or wrong parameter to 422, a refused operation to 403 and any other failure to 500, and there is no way yet to extend that with your own business errors.
- **Each way has its own error model.** If you may switch from one to the other later, wrap the calls in your own functions, because an HTTP exception and a bridge exception are different.

In short

Backbone gives a library one source of operations and a safe way to call them by id. Two ways into Python came out of that: an HTTP SDK, which is a client of a service, and a bridge, which is the library in your own process. The technical differences are real, but the one that decides for many is the license.

The Derafu Backbone page has an overview, and the reference of the Python bridge and of the dispatcher has every detail. LibreDTE explains why Lib Core is built on Backbone in this article, in Spanish, and it documents its bridge and its API.

Last updated on 15/11/2026