



Symfony Live  
PARIS 2018  
29-30 MARS



**Utiliser HTTPPlug Bundle en environnement de test.**

Présentation par Gary PEGEOT – Lead Dev @HighCo-Data

# Sommaire

- I. Présentation de HTTPPlug Bundle,
- II. Le mock-client,
- III. Le « FileClient »,
- IV. Le mot de la fin.



# I. HTTPPlug (et son bundle)

## De l'abstraction pour vos requêtes HTTP :

Une implémentation de **PSR-7** sur le modèle requête → réponse

(Semble familier ?)

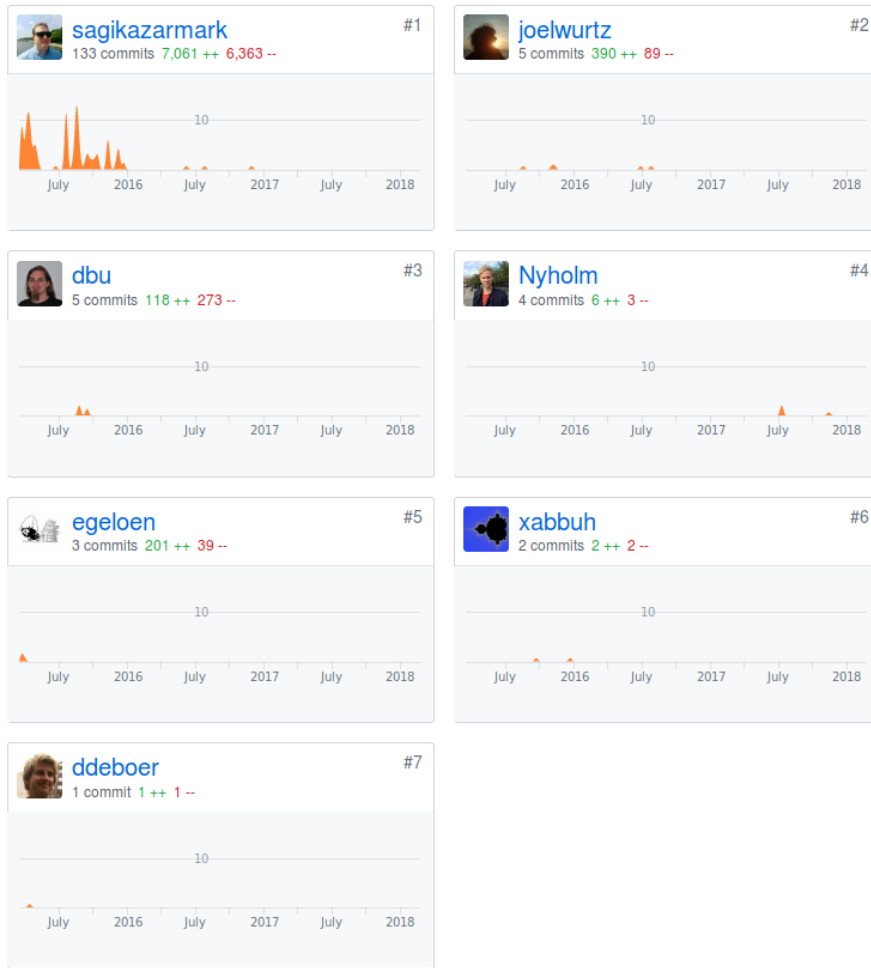
Compatible avec la proposition **PSR-18**,

Séparation de la logique métier de la librairie utilisée :

(Guzzle, ReactPHP, cURL,...)

Requêtes asynchrones

# 1. Fait par :



Première release en janvier 2016



SymfonyLive  
PARIS 2018  
29-30 MARS

# I. Qui l'utilise ?

Geocoder

## PHP Geocoder

Fetches geocode data from many different APIs.



## Mailgun

API client for Mailgun email service.

Fos  
Bundles

## FOSHttpCache

Integrate your application with HTTP caching proxy such as Varnish.



## KNPLabs Github client

An API client for GitHub.



## LinkedIn API client

An unofficial API client for communicating with LinkedIn API.



## Swap

A money currency converter library.



## Prooph ServiceBus Http Producer

Http producer for the prooph service-bus



## Akeneo PIM API client

PHP client of Akeneo PIM API



## PandaClient

The PandaClient package provides an easy to use implementation of the Panda encoding services REST API.



## Gitlab API client

A PHP wrapper to be used with Gitlab's API.



## Sentry

The official PHP SDK for Sentry (sentry.io)



## Auth0

Auth0-PHP SDK



## HWIOAuthBundle

OAuth client integration for Symfony.



SymfonyLive  
PARIS 2018  
29-30 MARS

# I. Installation

Symfony Flex / Pas Flex ?



SymfonyLive  
**PARIS 2018**  
**29-30 MARS**

# I. Installation

## Pas Flex :



```
composer require php-http/httpplug-bundle [some-adapter?]
```



```
public function registerBundles()  
{  
    $bundles = array(  
        // ...  
        new Http\HttpplugBundle\HttpplugBundle(),  
    );  
}
```



# I. Configuration de HTTPPlug

## Configuration :

```
httpplug:  
  plugins:  
    logger: ~  
  clients:  
    acme:  
      factory: 'httpplug.factory.guzzle6'  
      plugins: ['httpplug.plugin.logger']  
      config:  
        timeout: 2
```

## Usage :

```
$request = $this->container->get('httpplug.message_factory')->createRequest('GET', 'http://example.com');  
$response = $this->container->get('httpplug.client.acme')->sendRequest($request);
```



# I. Installation

## Flex :



```
composer require http
```

## Installation par défaut :

Client cURL choisi pour vous

Configuration d'un client avec différents plugins



# I. Pour résumer

## Les avantages de HTTPPlug Bundle :

Logique métier découplée du client utilisé :

- Respect du principe d'inversion des dépendences

- Code plus SOLID

- Simplicité d'utilisation

- Maintenance facilitée

- Testabilité



# I. Tester, c'est douter.

## Quelques bonnes raisons de « mock » ses web services :

Isolation des tests

Risque de DoS

Performances

Sécurité

Testabilité

## II. Utiliser le mock-client

### Cas pratique - Authentification sur une API externe (Magento) :

Vérification de la validité des identifiants et récupération des informations utilisateurs

#### Étapes :

1. Installer le mock-client & configurer,
2. Tester !

## II. Installer le mock-client

Installation :



```
composer require php-http/mock-client --dev
```

## II. Tester avec le mock-client

### Avec le bundle :

Client déclaré en temps que service : [httpplug.client.mock](#)

Factory disponible : [httpplug.factory.mock](#)

## II. Configurer le mock-client

### Configuration :

```
# config_test.yml
httpplug:
  clients:
    niafango:
      factory: 'httpplug.factory.mock' # replace factory
```

## II. Tester avec le mock-client

### Fonctionnement de base :

Ajouter des réponses

Ajouter des exceptions

Comportement par défaut



## II. La classe à tester

```
1 public function authenticate(string $username, string $password): bool
2 {
3     $request = $this->factory->createRequest(
4         'POST',
5         '/rest/V1/integration/customer/token',
6         ['Content-Type' => 'application/json', 'Accept' => 'application/json'],
7         \json_encode(\compact('username', 'password'))
8     );
9
10    $response = $this->client->sendRequest($request);
11
12    if (200 === $response->getStatusCode()) {
13        $this->token = \json_decode((string) $response->getBody(), true);
14
15        return true;
16    }
17
18    // Call the police
19 }
```

## II. Le test associé

### Étapes :

1. Créer un « mock » de réponse,
2. Ajouter la / les réponse au client Http\Mock\Client.

```
1 /** @var \PHPUnit\Framework\MockObject\MockObject|ResponseInterface $response */  
2 $response = $this->createMock(ResponseInterface::class);  
3 $response->method('getStatusCode')->willReturn(200);  
4 $response->method('getBody')->willReturn('"a1bearer1t0ken"');
```

## II. Le test associé

### Ajout de la réponse au client :

(Depuis un WebTestCase de Symfony)

```
1 $client = static::createClient();
2 $client->enableProfiler();
3 // $client->disableReboot(); → May save your keyboard life.
4
5 // httpplug.client.mock → Http\Mock\Client
6 $client->getContainer()->get('httpplug.client.mock')->addResponse($response);
7
8 $client->request('POST', '/connexion', ['_email' => 'miaou.lechat@highco-data.fr', '_password' => 'meow']);
9
10 /** @var \Symfony\Bundle\SecurityBundle\DataCollector\SecurityDataCollector $collector */
11 $collector = $client->getProfile()->getCollector('security');
12 $this->assertTrue($collector->isAuthenticated(), 'User should be authenticated.');
```

## II. Jusqu'ici, tout va bien

### Une méthode vite limitée :

Nécessite une action manuelle,

Test fortement couplés à l'implémentation (ordre des appels),

Difficile à utiliser à plusieurs endroits,

Lisibilité si beaucoup de contenu,

Pas forcément très pratique lorsqu'il y a beaucoup de requêtes.

# III. Un peu d'automatisation avec le « File Client »

## File Client - Le concept :

Transformer une requête en chemin, puis récupérer le fichier correspondant :

Ex : <https://example.org/foo/bar.json> →  
tests/Resources/HTTP/example.org/foo\_bar.json

Différents niveaux de granularité :

Préfixer par la méthode HTTP

Générer une signature :

*Headers,*

*Body,*

*Uri*



# III. La signature

## Génération de la signature :

Une requête → Un fichier :

Éviter les paramètres aléatoires :

Nonce dans la query,

JWT avec un iat,

Date dans le corps de la requête...



# III. Le code associé

```
1 public function generate(RequestInterface $request): string
2 {
3     $pieces = [
4         $request->getMethod(),
5         $request->getUri()->getHost(),
6         $request->getUri()->getPath(),
7     ];
8
9     if ($this->useQuery) {
10         $pieces[] = $request->getUri()->getQuery();
11     }
12
13     if ($this->useHeaders) {
14         foreach ($request->getHeaders() as $name => $values) {
15             $pieces[] = "$name: " . implode(' ', $values);
16         }
17     }
18
19     if ($this->useBody) {
20         $pieces[] = (string) $request->getBody();
21     }
22
23     return \md5(\implode('', $pieces));
24 }
```



### III. Une interface pour les gouverner tous.

Pour les requêtes synchrones :

**Http\Client\HttpClient**

Pour les requêtes asynchrones :

**Http\Client\HttpAsyncClient**



# III. À l'intérieur du client

```
protected function createResponse(RequestInterface $request): ResponseInterface
{
    $host = $request→getUri()→getHost();
    $path = \str_replace('/', '_', \trim($request→getUri()→getPath(), '/'));
    $method = \strtoupper($request→getMethod());
    $filesystem = new Filesystem();

    $directory = "{$this→dir}/$host/$path/$method";

    if (!$filesystem→exists($directory)) {
        $filesystem→mkdir($directory);
    }

    $filename = "$directory/{$this→generator→generate($request)}";

    if (!$filesystem→exists($filename)) {
        throw new \Exception("Please create '$filename' file.");
    }

    return $this→factory→createResponse(200, null, $this→headers, \file_get_contents($filename));
}
```

# III. Utiliser le FileClient

## Configuration en fonction du besoin

Pour un seul client HTTP :

- Déclarer le **FileClient** en temps que **service**
- Remplacer le client dans **config\_test.yaml** (SF 2/3) ou **services\_tests.yaml**.

## III. Remplacer le client

```
1 # services_test.yaml / config_test.yaml
2
3 services:
4     App\Tests\HTTP\FileClient:
5         $directory: '%kernel.project_dir%/some/dir'
6
7     App\HTTP\MyAwesomeWebService:
8         $client: '@App\Tests\HTTP\FileClient'
```

## III. L'usine

### Une configuration personnalisée

Plusieurs clients avec configuration unique :

- Créer une factory paramétrable
- Configurer chaque client indépendamment
- Possibilité d'utiliser plusieurs dossier de fixtures
- Définition par client des paramètres de la signature

# III. La configuration de test (ou de dev)

## Configuration souhaitée:

Factory avec différentes options par clients :

```
# config_test.yaml

httpplug:
  clients:
    foo:
      factory: 'httpplug.factory.file'
      config:
        file_directory: '%kernel.project_dir%/tests/Resources/HTTP/Foo'
        use_query: true
    bar:
      factory: 'httpplug.factory.file'
      config:
        file_directory: '%kernel.project_dir%/tests/Resources/HTTP/Bar'
        use_headers: false
```

# III. L'usine

```
1 use Http\HttplugBundle\ClientFactory\ClientFactory;
2 use Symfony\Component\OptionsResolver\OptionsResolver;
3
4 class FileClientFactory implements ClientFactory
5 {
6     //...
7     public function createClient(array $config = [])
8     {
9         $resolver = new OptionsResolver();
10        $resolver->setDefaults(['use_query' => true, 'use_body' => true, 'use_headers' => true]);
11
12        $resolver->setRequired('file_directory');
13        $resolver->setAllowedTypes('file_directory', 'string');
14        $options = $resolver->resolve($config);
15
16        return new FileClient(
17            $this->factory,
18            new SignatureGenerator($options['use_query'], $options['use_body'], $options['use_headers']),
19            $options['file_directory']
20        );
21    }
22 }
23
```

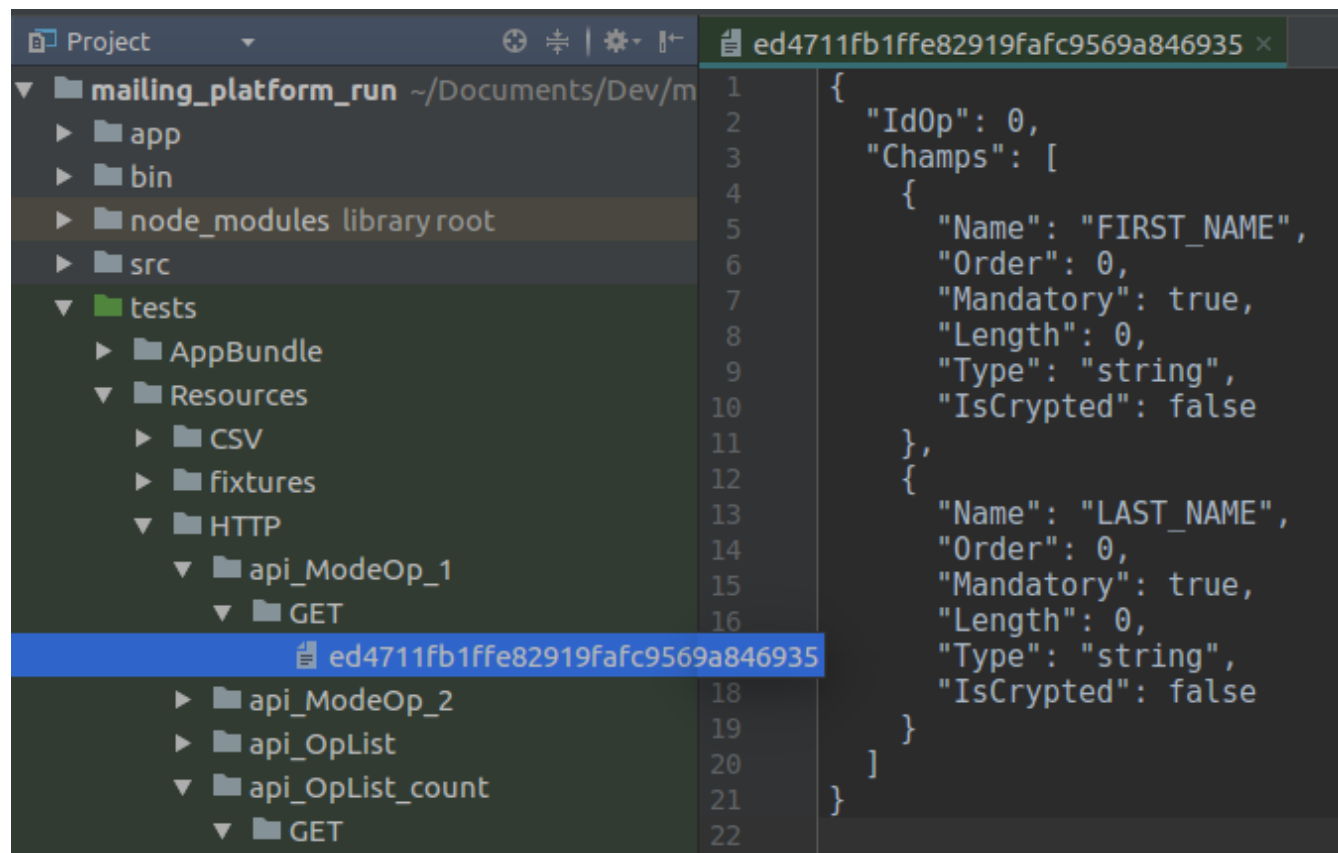
# III. Les premières fois...

## Premier lancement :

```
PHPUnit 5.7.27 by Sebastian Bergmann and contributors.  
  
Testing Project Test Suite  
.E  
  
Time: 343 ms, Memory: 28.00MB  
  
There was 1 error:  
  
1) Tests\AppBundle\Command\FetchOperationsCommandTest::testExecute  
Exception: Please create '/tests/Resources/HTTP//api_OpList/GET/67119b2046fdfe4eccb8ebf0f7df0124' file.
```

# III. La création des fixtures

L'arborescence des fichiers & résultats attendus :



The screenshot shows an IDE with a file explorer on the left and a code editor on the right. The file explorer displays the project structure for 'mailing\_platform\_run'. The 'tests' directory is expanded, showing 'AppBundle', 'Resources', and 'HTTP'. Under 'HTTP', 'api\_ModeOp\_1' is expanded, and 'GET' is selected. The code editor shows the JSON content of the selected file, which is a fixture for a GET request. The JSON structure includes an 'IdOp' field, a 'Champs' array with two objects for 'FIRST\_NAME' and 'LAST\_NAME', and a closing brace for the fixture.

```
Project
└─ mailing_platform_run ~/Documents/Dev/m
   └─ tests
      └─ HTTP
         └─ api_ModeOp_1
            └─ GET
               ed4711fb1ffe82919fafc9569a846935
                  {
                    "IdOp": 0,
                    "Champs": [
                      {
                        "Name": "FIRST_NAME",
                        "Order": 0,
                        "Mandatory": true,
                        "Length": 0,
                        "Type": "string",
                        "IsCrypted": false
                      },
                      {
                        "Name": "LAST_NAME",
                        "Order": 0,
                        "Mandatory": true,
                        "Length": 0,
                        "Type": "string",
                        "IsCrypted": false
                      }
                    ]
                  }
            └─ api_ModeOp_2
            └─ api_OpList
            └─ api_OpList_count
               └─ GET
```



## III. Tout va bien

Une fois le fichier complété :

```
PHPUnit 5.7.27 by Sebastian Bergmann and contributors.  
  
Testing Project Test Suite  
.....  
  
Time: 2.09 seconds, Memory: 52.00MB  
  
OK (32 tests, 93 assertions)
```

## IV. Le mot de la fin

### Avantages du mock client :

Pratique si peu de requêtes,  
Contrôle de l'ordre d'appel,  
Test des « edge cases ».

### Avantages du file client :

Gain de temps,  
Lisibilité des tests,  
Utilisable en dev.





Symphony Live  
PARIS 2018  
29-30 MARS



**Merci de votre attention !**

@GPegeot