PHP Workers via PHP-FPM

Ilia Alshanetsky

@iliaa - ilia@ilia.ws

Me, myself and I ;-)

- CTO @ Silofit – We are Hiring!!
- PHP Core Contributor & Ex-Release Master
- Author & Co-Author of multiple PHP extensions
- Security Nerd, wrote Guide to PHP Security
- Fascinated by making things faster
- Occasional Photographer

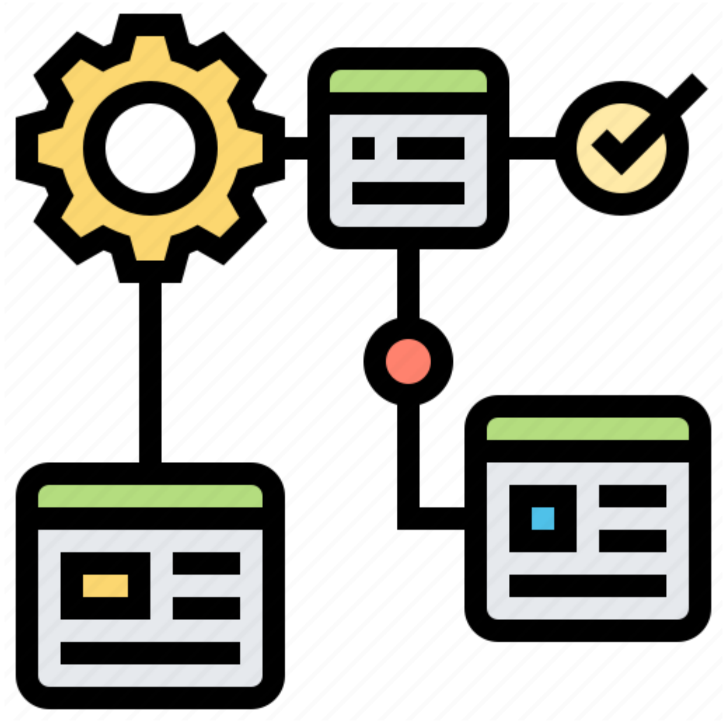
What is PHP-FPM?

Common Gateway Interface

1. PHP - FastCGI Process Manager (FPM)
2. The “thingy” helping your webserver understand & process PHP
3. Worker pool of PHP workers (processes)
4. Facilitates resource pooling (ie. Opcode Cache)
5. Fast, Stable, Configurable

Background, Batch & Parallel Processing

1. **Reliable**
2. **Easy to use**
3. **Scalability**
4. **Predictable (when things go wrong)**
5. **Instrumentable**



Reasons for PHP-FPM

1. **No warm-up time**
2. **Shared Opcode Cache**
3. **Availability of persistent connections**
4. **Auto-scale up / down depending on load**
5. **Stable & Proven**
6. **Configurable**
7. **Real-time metrics & slow script logging**
8. **Full process isolation**
9. **Real-time code refresh**

Reasons against PHP-FPM

1. **Not plug & play, yet**
2. **Not distributed out-of-the-box**

Client Library

https://github.com/hollodotme/fast-cgi-client

```
composer require hollodotme/fast-cgi-client
```

1. **Supports PHP 7.1 – 8.1**
2. **Well maintained**
3. **Easy API**
4. **Great documentation!**

Making the Connection

TCP

```php
use hollodotme\FastCGI\SocketConnections\NetworkSocket;

$socket = new NetworkSocket(
    'localhost',    # Hostname
    9000,           # Port
    2000,           # Connect timeout in milliseconds (default: 5000)
    1000            # Read/write timeout in milliseconds (default: 5000)
);
```

UDS

```php
use hollodotme\FastCGI\SocketConnections\UnixDomainSocket;

$socket = new UnixDomainSocket(
    '/var/run/php/php7.4-fpm.sock',    # Socket path
    1000,                              # Connect timeout in milliseconds (default: 5000)
    1000                               # Read/write timeout in milliseconds (default: 5000)
);
```

Sending Request

```
use hollodotme\FastCGI\Client;
use hollodotme\FastCGI\Requests\PostRequest;
use hollodotme\FastCGI\SocketConnections\NetworkSocket;

$client     = new Client();
$socket     = new NetworkSocket('localhost', 9000);
$payload    = http_build_query(['key' => 'value']);
$request    = new PostRequest('/path/to/script.php', $payload);

$response = $client->sendRequest($socket, $request);
```

$_POST payload to send to target

Full path to the script to execute

Output data available from response object

Handling Responses

```
setcookie("cookie", "monster");
echo "Hello World!";
```

Our test script

```
# Get all header values as an array of headers
$response->getHeader('Set-Cookie');
# ['cookie=monster']

# Get header as a string
$response->getHeaderLine('Set-Cookie');
# cookie=monster

# Get all headers as an associated array
$response->getHeaders();
# [
#   'Set-Cookie'   => ['cookie=monster'],
#   'Content-type' => ['text/html; charset=UTF-8'],
# ]
```

```
# Get response body
$response->getBody(); // Hello World!

# Get the raw response (headers + body)
$response->getOutput();
# Set-Cookie: cookie=monster
# Content-type: text/html; charset=UTF-8
#
# Hello World
```

```
# Get STDERR output
$response->getError();

# Get request duration
$response->getDuration(); // 0.001233
```

Only on critical errors (ie. Script not found)

Fire & Forget

```php
use hollodotme\FastCGI\Client;
use hollodotme\FastCGI\Requests\PostRequest;
use hollodotme\FastCGI\SocketConnections\NetworkSocket;

$client     = new Client();
$socket     = new NetworkSocket('localhost', 9000);
$payload    = http_build_query(['key' => 'value']);
$request    = new PostRequest('/path/to/script.php', $payload);

$socket_id = $client->sendAsyncRequest($socket, $request);

echo "Request sent, got ID: {$socket_id}, my work here is done!";
```

Uhm… so what happened?

```
# Blocking call until response is received or read timed out
$response = $client->readResponse($socket_id, 1000);
# wait up-to 1s to get a response
# timeout is an optional param, defaults to socket settings

echo $response->getBody();
```

Blocking call

Callbacks & Impatience

```
# Success callback
$request->addResponseCallbacks(
    static function( ProvidesResponseData $response ) {
        /* ... */
    }
);

# Failure callback
$request->addFailureCallbacks(
    static function ( Throwable $throwable ) {
        /* ... */
    }
);
```

```
while(1) {
    # Are we there yet?!
    if ($client->hasResponse($socket_id)) {
        $client->handleResponse($socket_id, 1000);
        break;
    }
    # Let's not kill the CPU
    usleep(100000);
}
```

Multiplicity

```php
$request1 = new PostRequest('/path/to/script.php', http_build_query(['val' => '1']));
$request2 = new PostRequest('/path/to/script.php', http_build_query(['val' => '2']));
$request3 = new PostRequest('/path/to/script.php', http_build_query(['val' => '3']));

$socket_ids = [
    $client->sendAsyncRequest($socket, $request1),
    $client->sendAsyncRequest($socket, $request2),
    $client->sendAsyncRequest($socket, $request3),
];


# Read all responses, blocking until all responses are received
# which will be returned in the order executed
foreach ($client->readResponses(2000, ...$socket_ids) as $response)
{
   # …
}
```

Reactive Approach

```
while ( $client->hasUnhandledResponses() )  {

    $ready_sockets = $client->getSocketIdsHavingResponse();

    # process data from all the ready sockets
    foreach ( $ready_sockets as $socket_id ) {
        $response = $client->readResponse($socket_id, 1000);
    }

    usleep(100000); # 0.1 second wait
}
```

```
$client->handleResponse($socket_id, 1000);
```

When using callbacks, simply change `readResponse` **to** `handleResponse`

Putting it all together

```php
$running = 0;
while (1) {
    # Allow up-to 10 running process, while there are new tasks
    while ( $running < 10 && $task = getTaskFromQueue() ) {
        $client->sendAsyncRequest(
            $socket,
            new PostRequest($task['script'], http_build_query($task['data']))
        );
        ++$running; # increase process counter
    }

    if ( $client->hasUnhandledResponses() )  {
        $ready_sockets = $client->getSocketIdsHavingResponse();

        foreach ( $ready_sockets as $socket_id ) {
            $response = $client->readResponse($socket_id, 1000);
            processResponse($response);
            --$running; # decrease task counter
        }
    }

    usleep(50000); # give CPU 0.05 second break between cycles
}
```

Other Request Types

GET	`hollodotme\FastCGI\Requests\GetRequest`
PATCH	`hollodotme\FastCGI\Requests\PatchRequest`
DELETE	`hollodotme\FastCGI\Requests\DeleteRequest`
PUT	`hollodotme\FastCGI\Requests\PutRequest`

Same input as POST

Generally, POST is all your need

```php
use hollodotme\FastCGI\RequestContents\JsonData;

$json = new JsonData([
    'data' => [
        1,
        'two'   => 'value',
        'array' => [
            'value2',
        ],
    ],
]);

$request = PostRequest::newWithRequestContent(
    '/path/to/script.php',
    $json
);
```

```php
use hollodotme\FastCGI\RequestContents\MultipartFormData;

$payload = new MultipartFormData(
    # POST data
    [ 'key' => 'data'],
    # FILES
    ['file' => '/path/to/file']
);

$request = PostRequest::newWithRequestContent(
    '/path/to/script.php',
    $payload
);
```

Thank you for listening!