Async HTTP Retriever Documentation

Jon Cram

Dec 19, 2018

First Steps

Service for retrieving HTTP resources asynchronously. Self-hosted within a lovely collection of docker containers.

Send a `POST` request containing `url`, `callback`, (optionally) `header` and (optionally) `parameter` values. Content for the given `url` will be retrieved *eventually* and sent in a `POST` request to the specified `callback` url.

Everything becomes clear if you read the *overview*. There is even a handy diagram.

CHAPTER 1

Requirements

You'll need `docker` and `docker-compose` present on the host box. If you can run docker, you can run this application. Nothing else needed.

Developed and tested against `docker` 18.06.1-ce and `docker-compose` 1.22.0.

CHAPTER 2

First Steps

Have a look at the *getting started guide* for details on how to *get the code*, *create your configuration* and *install an instance*.

2.1 Overview

Service for retrieving HTTP resources asynchronously. Self-hosted within a lovely collection of docker containers.

2.1.1 Short Description

Send a `POST` request containing `url`, `callback`, (optionally) `header` and (optionally) `parameter` values. Content for the given `url` will be retrieved *eventually* and sent in a `POST` request to the specified `callback` url.

```
+---------------+                                                        +----------
↪-----+
|               |                                                        |         ␣
↪     |
|               | POST http://localhost:8001/                            |         ␣
↪     |
|               | url=http://example.com/                                |         ␣
↪     |
|               | callback=http://callback.example.com/                  |         ␣
↪     |
|               | headers={                                              |         ␣
↪     |
|               |     "User-Agent":"Chrome, honest"                      |         ␣
↪     |
|               | }                                                      |         ␣
↪     |
|               | parameters={                                           |         ␣
↪     |
|               |     "cookies": {                                       |         ␣
↪     |
```

(continues on next page)

(continued from previous page)

```
|              |     }                                                      |        ␣
↪      |
|              |   }                                                          |        ␣
↪      |
|              |                                                            |        ␣
↪      |
|              |                                                            |        ␣
↪      |
|              |                                                            |␣
↪Asynchronous  |
| Your         |                                                            | HTTP    ␣
↪      |
| application  | +---------------------------------------------------------> |␣
↪retriever     |
|              |                                                            |        ␣
↪      |
|              |                                                            |        ␣
↪      |
|              |                                                            |        ␣
↪      |
|              |                                                            |        ␣
↪      |
|              |                       HTTP 200 OK                          |        ␣
↪      |
|              |                       Content-Type: application/json       |        ␣
↪      |
|              |                                                            |        ␣
↪      |
|              |                       "118e35f631be802c41bec5c9dfb0f415"   |        ␣
↪      |
|              | <---------------------------------------------------------+ |        ␣
↪      |
+--------------+                                                            +---------
↪-----+
+
```

… some time passes …

```
+-------------+                                                             +---------
↪-----+
|             |                    POST http://callback.example.com/        |        ␣
↪      |
|             |                    {                                        |        ␣
↪      |
|             |                      "request_id": "118e35f631be802c41b...", |        ␣
↪        |
|             |                      "status": "success",                   |        ␣
↪      |
|             |                      "headers": {                           |        ␣
↪      |
|             |                        "content-type": "text/html;"         |        ␣
↪      |
| Your        |                      },                                     |        ␣
↪      |
| callback    |                      "content": "PGRvY3R5cGUgaHRtbD4="      |␣
↪Asynchronous |
| handler     |                    }                                        | HTTP    ␣
↪      |
```

(continues on next page)

(continued from previous page)

```
|              |                                                          |␣
↪Retriever     |
|              | <--------------------------------------------------------+ |         ␣
↪      |
+--------------+                                                          +----------
↪-----+
```

2.1.2 Why?

Pretty much every modern programming ecosystem provides a means for making HTTP requests and handling the resulting responses. You already get synchronous HTTP out the box, possibly asynchronous HTTP as well. Using whatever HTTP functionality your programming ecosystem provides is fine most of the time.

Want to retrieve the content of arbitrary urls often? No, you probably don't. But if you do, you periodically run into failure cases.

We don't like failure cases. Temporary service unavailability, intermittent internal server errors, unpredictable rate limiting responses.

To reliably retrieve an arbitrary HTTP resource, you need to able to retry after a given period for those odd cases where a request failed *right now* but which could (maybe would) succeed *a little later*. You introduce state (remembering *what* to retrieve) and you need something to handle doing so *at the right time* (some form of delayable background job processing).

You could re-write the means for doing so for every application you create that needs to retrieve resources over HTTP. Or you could not. Up to you really.

2.1.3 Production Readiness

Not production ready

2.2 Getting Started

2.2.1 Requirements

You'll need `docker` and `docker-compose` present on the host box. If you can run docker, you can run this application. Nothing else needed.

Developed and tested against `docker` 18.06.1-ce and `docker-compose` 1.22.0.

2.2.2 Getting the Code

```
git clone git@github.com:webignition/async-http-retriever.git
```

2.2.3 Creating Your Configuration

Configuration is provided through a collection of environment variables. These can be set on the host itself or defined in `docker/.env`. I'm assuming the use of a `.env` file to be consumed by `docker-compose`.

There are some configuration values you must set before installing. There are some configuration values that can only be set before installing.

```
cd /var/www/async-http-retriever/instance-x
cp docker/.env.dist docker/.env
```

Edit your `docker/.env` file as needed. Refer to the *configuration guide*

2.2.4 Installation

You've got the code and you've set your configuration via environment variables. Now to install.

The following *simple installation guide* briefly covers how to install an instance.

If you ever want to perform zero-downtime upgrades (yes, yes, do you) or if you ever want to run multiple instances on the same host, you want to create an *isolated installation*.

Simple Installation

```
# Create a directory for the application and get the code
mkdir -p /var/www/async-http-retriever/instance-x
cd /var/www/async-http-retriever/instance-x
git clone git@github.com:webignition/async-http-retriever.git .

# Change to the docker directory as that is where all the fun happens
cd docker

# Build container images
docker-compose up -d --build

# Install third-party dependencies
docker-compose exec -T app-web ./bin/console composer install

# Create database schema
docker-compose exec -T app-web ./bin/console doctrine:migrations:migrate --no-
↪interaction

# Restart containers
# (will have failed due to lack of third-party dependencies and lack of database␣
↪schema)
docker-compose down && docker-compose up -d
```

Isolated Installation

An isolated instance:

- has container names unique to itself
- has service port numbers unique to itself
- will not interfere with other instances on the same host
- will not be interfered by other instances on the same host

Isolated installations are preferred.

We can isolate instances through the use of:

- the docker-compose project name argument
- the `ID` environment variable
- the `ASYNC_HTTP_RETRIEVER_EXPOSED_PORT` environment variable
- the `ASYNC_HTTP_RETRIEVER_RABBITMQ_MANAGEMENT_EXPOSED_PORT` environment variable

An isolated instance has a name. This can be whatever is meaningful to you. For this example, we'll opt for the name `instance-x`.

```
##########
# GET CODE
##########

# Create a directory for the application and get the code
mkdir -p /var/www/async-http-retriever/instance-x
cd /var/www/async-http-retriever/instance-x
git clone git@github.com:webignition/async-http-retriever.git .

# Create a directory for the MySQL data files for this instance
mkdir -p /var/docker-mysql/async-http-retriever-x

# Change to the docker directory as that is where all the fun happens
cd docker

# CONFIGURE

# Create the configuration
cp .env.dist .env

# We need to set the MySQL data path on the host
sed -i 's|MYSQL_DATA_PATH_ON_HOST|/var/docker-mysql/async-http-retriever-x|g' .env

# Set a non-default rabbit-mq management interface port
sed -i 's|15672|25672|g' .env

# Set a non-default application port
sed -i 's|8001|8002|g' .env

# INSTALL

ID=instance-x docker-compose -p instance-x up -d --build
ID=instance-x docker-compose -p instance-x exec -T app-web composer install
ID=instance-x docker-compose -p instance-x exec -T app-web ./bin/console␣
↪doctrine:migrations:migrate --no-interaction
ID=instance-x docker-compose -p instance-x down
ID=instance-x docker-compose -p instance-x up -d
```

You must pass in the `ID` environment variable and the project name when calling `docker-compose` commands:

```
# List containers
ID=instance-x docker-compose -p instance-x ps

# ssh into the app-web container
ID=instance-x docker-compose -p instance-x exec app-web /bin/bash
```

2.3 Request-Response Cycle

Your application *requests a resource*.

This request includes the `url` of the resource you want to retrieve, the `callback` URL where you want the resource to be sent when it has been retrieved and, optionally, a set of `headers` to be sent with the retrieval request.

The response you receive includes a json-encoded string. That's the *request ID*. You'll want to make a note of that somewhere.

```
+---------------+                                                            +---------
↪-----+
|               |                                                            |        ␣
↪      |
|               | POST http://localhost:8001/                                |        ␣
↪      |
|               | url=http://example.com/                                    |        ␣
↪      |
|               | callback=http://callback.example.com/                      |        ␣
↪      |
|               | headers={                                                  |        ␣
↪      |
|               |     "User-Agent":"Chrome, honest"                          |        ␣
↪      |
|               | }                                                          |        ␣
↪      |
|               | parameters={                                               |        ␣
↪      |
|               |     "cookies": {                                           |        ␣
↪      |
|               |     }                                                      |        ␣
↪      |
|               | }                                                          |        ␣
↪      |
|               |                                                            |        ␣
↪      |
|               |                                                            |        ␣
↪      |
|               |                                                            |␣
↪Asynchronous  |
| Your          |                                                            | HTTP   ␣
↪      |
| application   | +--------------------------------------------------------> |␣
↪retriever     |
|               |                                                            |        ␣
↪      |
|               |                                                            |        ␣
↪      |
|               |                                                            |        ␣
↪      |
|               |                                                            |        ␣
↪      |
|               |                      HTTP 200 OK                           |        ␣
↪      |
|               |                      Content-Type: application/json        |        ␣
↪      |
|               |                                                            |        ␣
↪      |
```

(continues on next page)

(continued from previous page)

```
|              |                      "118e35f631be802c41bec5c9dfb0f415"   |        ␣
↪     |
|              | <-------------------------------------------------------+ |        ␣
↪     |
+--------------+                                                         +---------
↪-----+
+
```

Your request to retrieve a resource has been put into a queue. The request will probably be handled quite quickly but not instantly. Some time will pass before your request has completed.

. . . let's wait. Something will happen eventually . . .

Your request completed and was successful. That's good.

A json-encoded *response object* is sent in a `POST` request to the `callback` URL that you specified in your request.

```
+-------------+                                                          +---------
↪-----+
|             |                    POST http://callback.example.com/     |        ␣
↪     |
|             |                    {                                     |        ␣
↪     |
|             |                      "request_id": "118e35f631be802c41b...", |    ␣
↪       |
|             |                      "status": "success",                |        ␣
↪     |
|             |                      "headers": {                        |        ␣
↪     |
|             |                        "content-type": "text/html;"      |        ␣
↪     |
| Your        |                      },                                  |        ␣
↪     |
| callback    |                      "content": "PGRvY3R5cGUgaHRtbD4="   |␣
↪Asynchronous |
| handler     |                    }                                     | HTTP   ␣
↪     |
|             |                                                          |␣
↪Retriever    |
|             | <--------------------------------------------------------+ |      ␣
↪     |
+-------------+                                                          +---------
↪-----+
```

2.4 Requesting a Resource

2.4.1 Making a Request

Parameters

Send a `POST` request to your instance. Include the `url` of the resource to be retrieved and the `callback` URL to where the response should be sent.

You can optionally provide a `headers` parameter defining headers to send when retrieving the resource.

The collection of headers can contain whatever keys and values you need to satisfy a request. This might include specifying the User-Agent, passing along Authorization or setting Cookies.

Name	Description	Example
`url`	URL of the resource to be retrieved	`http://example.com`
`callback`	URL to which the resource should be sent	`https://httpbin.org/post`
`headers`	JSON-encoded key:value pairs	`{"User-Agent":"Chrome, honest"}`
`parameters`	JSON-encoded parameters	`{"cookies":{"domain": "..."}}`

Curl Example Without Headers

```
curl -X POST http://localhost:8001/ \
     -d 'url=http://example.com/&callback=https://httpbin.org/post'

"118e35f631be802c41bec5c9dfb0f415"
```

Curl Example With Headers

```
curl -X POST http://localhost:8001/ \
     -d 'url=http://example.com/&callback=https://httpbin.org/post&headers={"User-
↪Agent":"Chrome"}'

"ea8a4d4eb1840d0bec6284658a8ef064"
```

2.4.2 Specifying Cookie Parameters

Including a `Cookie` header in your request for a resource will result in an equivalent `Cookie` header being sent with the relevant HTTP request.

```
curl -X POST http://localhost:8001/ \
     -d 'url=http://example.com/&headers={"Cookie":"key=value"}&callback=https://
↪httpbin.org/post'
```

Cookies may contain sensitive information. The request for a resource may be redirected to another host. You do not want to pass potentially-sensitive information to another host. No, you don't. Trust me.

Add to your `parameters` value a cookie parameters object:

```
{
  "cookie": {
    "domain": ".example.com",
    "path": "/"
  }
}
```

```
curl -X POST http://localhost:8001/ \
     -d 'url=http://example.com/&headers={"Cookie":"key=value"}&parameters={"cookies":
↪{"domain":".example.com","path":"/"}}&callback=https://httpbin.org/post'
```

You must include `domain` and `path` values. It is up to you to choose the correct values for the resource you are requesting.

Only requests against URLs that match the given `domain` and `path` values will have the relevant `Cookie` header set. Cookie parameters prevent cookie data from being exposed where it should not be.

If you specify a `Cookie` header but do not specify cookie parameters, no cookies will be sent with the request to retrieve the resource.

2.4.3 Understanding The Response

Successful Request (200)

The response body (`"118e35f631be802c41bec5c9dfb0f415"` in the very first example) is a json-encoded request ID.

The request ID is unique to the combination of `url`, `headers` and `parameters`.

Store the request ID in *your* application. The request ID is sent with the requested resource to the given `callback` URL. Use the request ID to map the response you receive to the request that you made.

Bad Request (400)

Your request will receive a `HTTP 400` response if:

- `url` is empty
- `callback` is empty
- `callback` is not valid (which depends on your configuration for allowed callback host names)

2.5 Callback Responses

A request to retrieve a resource will be followed up (*eventually*) by a `POST` request to the given `callback` URL.

The body of the request is a json-encoded response object.

2.5.1 Response Object Properties

Name	Description	Example
`request_id`	Unique request identifier	`118e35f631be802c41bec5c9dfb0f415`
`status`	Whether the resource could be retrieved	`success` or `failed`
`failure_type`	If `status=failed`	`http`, `curl` or `unknown`
`status_code`	If `status=failed` and (`failure_type=http` or `failure_type=curl`)	`failure_type=http`: `404`, `500` ... `failure_type=curl`: `6`, `28` ...
`context`	Array of additional information If `status=failed` and `failure_type=http` and `status_code=301`	
`headers`	Response headers if `status=success`	`{"content-type": "text/html"}`
`content`	Base64-encoded response body in cases where `status=success`	`PGRvY3R5cGUgaHRtbD4=`

2.5.2 Success Response Example

```
{
  "request_id": "118e35f631be802c41bec5c9dfb0f415",
  "status": "success",
  "headers": {
    "content-type": "text/html; charset=utf-8",
    "content-length": 40,
    "cache-control": "public, max-age=60"
  },
  "content": "PGRvY3R5cGUgaHRtbD48aHRtbD48Ym9keT48L2JvZHk+PC9odG1sPg=="
}
```

2.5.3 HTTP Failure Example (404 Not Found)

```
{
  "request_id": "118e35f631be802c41bec5c9dfb0f415",
```

(continues on next page)

(continued from previous page)

```
  "status": "failed",
  "failure_type": "http",
  "status_code": 404
}
```

2.5.4 HTTP Failure Example (301)

```
{
  "request_id": "118e35f631be802c41bec5c9dfb0f415",
  "status": "failed",
  "failure_type": "http",
  "status_code": 404,
  "context": {
    "too_many_redirects": true,
    "is_redirect_loop": true,
    "history": [
        "http://example.com",
        "http://example.com",
        "http://example.com",
        "http://example.com",
        "http://example.com"
    ]
  }
}
```

2.5.5 Curl Failure Example (Operation Timed Out)

```
{
  "request_id": "118e35f631be802c41bec5c9dfb0f415",
  "status": "failed",
  "failure_type": "curl",
  "status_code": 28
}
```

2.5.6 Unknown Failure Example

```
{
  "request_id": "118e35f631be802c41bec5c9dfb0f415",
  "status": "failed",
  "failure_type": "unknown"
}
```

2.6 Upgrading

Upgrading a live instance is not recommended. Doing so can put your instance into an odd state resulting in service-unavailability the applications making use of your instance.

Upgrading with zero downtime is achievable. We will do that.

2.6.1 Zero-downtime Upgrading

We can achieve zero downtime for applications that use an existing instance by creating a second instance and using that instead. The first instance can then be removed.

Pre-requisites:

- you have an existing *isolated instance* named `instance-x`.
- you have determined that an upgrade is needed

Scenario:

- create instance-y (second instance)
- configure instance-y and get it running
- configure relevant applications on the host to use instance-y
- remove instance-x

2.6.2 Creating the Second Instance (`instance-y`)

```
##########
# GET CODE
##########

# Create a directory for the application and get the code
mkdir -p /var/www/async-http-retriever/instance-y
cd /var/www/async-http-retriever/instance-y
git clone git@github.com:webignition/async-http-retriever.git .

# Create a directory for the MySQL data files for this instance
mkdir -p /var/docker-mysql/async-http-retriever-y

# Change to the docker directory as that is where all the fun happens
cd docker

###########
# CONFIGURE
###########

# Create the configuration
cp .env.dist .env

# We need to set the MySQL data path on the host
sed -i 's|MYSQL_DATA_PATH_ON_HOST|/var/docker-mysql/async-http-retriever-y|g' .env

# Set the rabbit-mq management port to not be the same as instance-x
sed -i 's|15672|35672|g' .env

# Set the application port to not be the same as instance-x
sed -i 's|8001|8003|g' .env

#########
# INSTALL
#########

ID=instance-y docker-compose -p instance-y up -d --build
```

(continues on next page)

(continued from previous page)

```
ID=instance-y docker-compose -p instance-y exec -T app-web composer install
ID=instance-y docker-compose -p instance-y exec -T app-web ./bin/console␣
↪doctrine:migrations:migrate --no-interaction
ID=instance-y docker-compose -p instance-y down
ID=instance-y docker-compose -p instance-y up -d
```

2.6.3 Configure Applications to Use `instance-y`

You managed to configure relevant applications to use `instance-x`. Do the same but for `instance-y`.

You configure all relevant applications to use `instance-y` and that, once done, no applications are using `instance-x`.

2.6.4 Removing the First Instance

```
# Change to the docker directory as that is where all the fun happens
cd /var/www/async-http-retriever/instance-x/docker

# Stop and remove containers
ID=instance-x docker-compose -p instance-x down

# Remove instance-x
cd /var/www/async-http-retriever
rm -rf cd /var/www/async-http-retriever/instance-x

# Remove MySQL data files
rm -rf /var/docker-mysql/async-http-retriever-x
```

2.7 Configuration

Configuration is provided through a collection of environment variables. These can be set on the host itself or defined in `docker/.env`.

2.7.1 Creating your configuration file

Copy the relevant `.env.dist` to `.env`.

```
cp docker/.env.dist docker/.env
```

2.7.2 Configuration You Must Set

Things are not going to work nicely if you don't set these.

`ASYNC_HTTP_RETRIEVER_DATABASE_DATA_PATH`
The path *on the host* for MySQL to store data.

Set this to any writable directory that already exists. Do not set this to `/var/lib/mysql` if your host is running a MySQL instance.

This must be set before installing.

2.7.3 Configuration You Should Set

Things will work if you don't set these, however setting is recommended as some of these values are sensitive.

`ASYNC_HTTP_RETRIEVER_EXPOSED_PORT`
Port to expose for the application. Set to any suitable unused port number.

`ASYNC_HTTP_RETRIEVER_MYSQL_ROOT_PASSWORD`
The root password for the MySQL instance. Set to any value and forget about it.

This must be set before installing if you want to set it.

`ASYNC_HTTP_RETRIEVER_DATABASE_USER`
DB user for the application to use. Set to any value and forget about it.

This must be set before installing if you want to set it.

`ASYNC_HTTP_RETRIEVER_DATABASE_PASSWORD`
DB password for the application to use. Set to any value and forget about it.

This must be set before installing if you want to set it.

`ASYNC_HTTP_RETRIEVER_RABBITMQ_USER`
Username for the rabbit-mq service. Set to any meaningful value.

This must be set before installing if you want to set it.

`ASYNC_HTTP_RETRIEVER_RABBITMQ_PASS`
Password for the rabbit-mq service. Set to any meaningful value.

This must be set before installing if you want to set it.

2.7.4 Configuration You Can Optionally Set

Set these if you like, things will work just fine if you don't.

`ASYNC_HTTP_RETRIEVER_CONSUMER_COUNT`
Number of parallel message consumers. Defaults to 1. Ideally set higher.

`ASYNC_HTTP_RETRIEVER_APP_SECRET`
Private token used within the application. Set to whatever you like.

`ASYNC_HTTP_RETRIEVER_CALLBACK_ALLOWED_HOSTS`
Used to limit the host names allowed in callback URLs. Defaults to * which allows all host names.

`ASYNC_HTTP_RETRIEVER_RETRIEVER_TIMEOUT_SECONDS`
Timeout in seconds for when retrieving HTTP resources. Defaults to *30* seconds.

Set to any positive integer. Set to 0 for no timeout (probably a bad idea).

`ASYNC_HTTP_RETRIEVER_DATABASE_NAME`
Name of the application database.

This must be set before installing if you want to set it.

`ASYNC_HTTP_RETRIEVER_DATABASE_USER`
DB user for the application.

This must be set before installing if you want to set it.

`ASYNC_HTTP_RETRIEVER_RABBITMQ_MANAGEMENT_EXPOSED_PORT`
Exposed port of the rabbit-mq management interface.

`ASYNC_HTTP_RETRIEVER_HTTPBIN_EXPOSED_PORT`
Port to expose for httpbin when using the dev configuration. Defaults to 7000.