Rhubarb Documentation

Release

Robert Allen

January 02, 2014

Contents

Contents:

CHAPTER 1

Installation

1.1 Composer

The recommended method of installation is via composer

```
composer require 'zircote/rhubarb=3.2-dev'
```

Depending on your selection of connectors you will also need to require or compile the appropriate extension or libraries.

Libraries can be included utilising the composer command

```
composer require 'predis/predis'
```

1.2 PECL AMQP

The Official PHP AMQP extension may be found at https://github.com/bkw/pecl-amqp-official as well as stubs and tests.

Installation via pecl

```
sudo pecl install amqp
```

To build the ext-amqp from source:

```
#!/bin/sh

git clone https://github.com/alanxz/rabbitmq-c
pushd rabbitmq-c
    git submodule init
    git submodule update
    mkdir bin-rabbitmq-c
    cd bin-rabbitmq-c
    cmake ..
    make
    sudo make install
popd
git clone https://github.com/bkw/pecl-amqp-official.git
```

```
pushd pecl-amqp-official
    phpize . && ./configure
    make && make test
    sudo make install
    sudo echo "[amqp]" > $(path_to_php_ini)/conf.d/amqp.ini
    sudo echo "extension=amqp.so" >> $(path_to_php_ini)/conf.d/amqp.ini
popd
```

CHAPTER 2

backends

2.1 redis

Rhubarb supports redis by way of predis/predis

2.1.1 Configuration

Simple Predis Config

```
<?php
return array(
    'broker' => array('type' => 'predis'),
    'result_store' => array('type' => 'predis'),
    'tasks' => array(array('name' => 'app.add'))
);
```

Predis supports a number of options listed below:

- scheme [string - default: tcp] [tcp, unix, http]
- host [string - default: 127.0.0.1]
- port [integer - default: 6379]
- path [string - default: not set]
- database [integer - default: not set]
- password [string - default: not set]
- connection_async [boolean - default: false]
- connection_persistent [boolean - default: false]
- connection_timeout [float - default: 5.0]
- read_write_timeout [float - default: not set]
- alias [string - default: not set]
- weight [integer - default: not set]
- iterable_multibulk [boolean - default: false]

- throw_errors [boolean - default: true]

Details on the configuration may be found at https://github.com/nrk/predis/wiki/Connection-Parameters

Example Usage

```
<?php
return array(
    'broker' => array(
        'connection' => 'tcp://localhost:6379?password=54321'
    ),
    'result_store' => array(
        'connection' => 'tcp://localhost:6370?database=0'
    )
);
```

2.2 AMQP

Rhubarb supports **AMQP** by way of ext-amqp

Note: Note that at this time the **ext-amqp** extension does not support *TLS*

2.2.1 Configuration

Simple AMQP Config

```
<?php
return array(
    'broker' => array('type' => 'phpamqp'),
    'result_store' => array('type' => 'phpamqp'),
    'tasks' => array(array('name' => 'app.add'))
);
```

AMQP supports a number of options listed below:

- host [string - default: localhost]
- port [integer - default: 5672]
- vhost [string - default: celery]
- login [string - default: guest]
- password [string - default: guest]
- write_timeout [integer - default: -1]
- read_timeout [integer - default: -1]

These options may be used as an associative array or an URI:

URI string Connection definition

```
<?php
return array(
    'broker' => array(
        'type' => 'phpamqp',
        'connection' => 'amqp://guest:guest@localhost:5372/celery?read_timeout=5&write_timeout=2
```

```
    ),
    'result_broker' => array(
        'type' => 'phpamqp',
        'connection' => 'amqp://guest:guest@localhost:5372/celery'
    )
);
```

Array Connection definition

```
<?php
return array(
    'broker' => array(
        'type' => 'phpamqp',
        'connection' => array(
            'host' => 'localhost',
            'port' => 5372,
            'vhost' => 'celery',
            'login' => 'guest',
            'password' => 'guest',
            'write_timeout' => 5,
            'read_timeout' => 2
        )
    ),
    'result_store' => array(
        'type' => 'phpamqp',
        'connection' => array(
            'host' => 'localhost',
            'port' => 5372,
            'vhost' => 'celery',
            'login' => 'guest',
            'password' => 'guest',
            'write_timeout' => 5,
            'read_timeout' => 2
        )
    ),
);
```

CHAPTER 3

Rhubarb

Examples of Celery Worker Execution From PHP

Send AsyncResult and Wait For Result

```
use Rhubarb\Rhubarb;
use Rhubarb\Task\Args\Python as PythonTask;

$config = include('configuration/predis.php');
$rhubarb = new Rhubarb($config);
$argsPython = new PythonTask(1, 2);

try {
    $result = $rhubarb->task('app.add')
        ->delay($argsPython, array())
        ->get();
} catch (\Rhubarb\Exception\TimeoutException $e) {
    /*
     * If the task result is not received within '10' seconds (default) a
     * '\Rhubarb\Exception\TimeoutException' is thrown.
     */
    echo $e->getMessage(), PHP_EOL;
}
```

Send task with kwargs

```
use Rhubarb\Rhubarb;
use Rhubarb\Task\Args\Python as PythonArgs;
use Rhubarb\Task\Args\Python\Kwargs;

$config = include('configuration/predis.php');
$rhubarb = new Rhubarb($config);

$kwargs = new Kwargs(array('arg3' => 'this is kwarg three'));
$kwargs['arg_1'] = 'my first arg';
$kwargs->arg2 = 'the second arg';

$args = new PythonArgs($kwargs);

$result = $rhubarb->task('app.add')
    ->delay($args)
    ->get();
```

Send task using an invokable signature

```
use Rhubarb\Rhubarb;
use Rhubarb\Task\Args\Python as PythonArgs;

$config = include('configuration/amqp.php');
$rhubarb = new Rhubarb($config);

$args = new PythonArgs(1, 2);

$signature = $rhubarb->task('app.add');
$result = $signature($args)->get();
```

Send task with a 60 second countdown header

```
use Rhubarb\Rhubarb;
use Rhubarb\Task\Args\Python as PythonArgs;

$config = include('configuration/predis.php');
$rhubarb = new Rhubarb($config);
$args = new PythonArgs(1, 2);
```



```
$rhubarb->task('app.add')
    ->delay($args, array(), array('countdown' => 60)); /* Task will execute in 60 seconds */
```


Send task using ETA header

CHAPTER 4

Tasks

4.1 Args

Arguments

Why is the Args pattern complicated? Because, in version 3.2 of Celery message protocol v.2 defines support for language based task routing. To facilitate this for future use and expansion it become necessary to define argument objects that are language specific.
While support for multiple argument formats is possible, current workers only support `Python` args.

Basic Args creation with factory

```
$args = \Rhubarb\Task\Args::newArgs(
    \Rhubarb\Task\Args\Python::LANG,
    2,
    2,
    \Rhubarb\Task\Args\Python\Kwargs::newKwargs(array('args1'=>'val1', 'arg2' => 'val2')
);
```

Explicit Python Args using star args and kwargs

```
$args = new \Rhubarb\Task\Args\Python(
    2,
    2,
    \Rhubarb\Task\Args\Python\Kwargs::newKwargs(array('args1'=>'val1', 'arg2' => 'val2')
);
```

Extended RhubarbTaskArgsPythonKwargs usage Note the object property and array access usage for flexibility.

```
$kwargs = new \Rhubarb\Task\Args\Python\Kwargs();
$kwargs->arg1 = 'val1';
$kwargs['arg2'] = 'val2';
$args = \Rhubarb\Task\Args::newArgs(null, $kwargs);
```

Creating your own Arg types

- Creating your own arg types is as simple as creating an object implementing the `\Rhubarb\Task\ArgsInterface`
- Registering it with the `\Rhubarb\Task\Args` object
- Fetching it with the factory

```
/**
 * Class PhpArgs
 *
 * An hypothetical example class of an Args class for PHP
 */
class PhpArgs implements \Rhubarb\Task\Args\ArgsInterface
{
    const LANG = 'php';
    /**
     * @return mixed
     */
    public function serialize()
    {
        // TODO: Implement serialize() method.
    }

    /**
     * @return string
     */
    public function __toString()
    {
        // TODO: Implement __toString() method.
    }

    /**
     * @return array
     */
    public function toArray()
    {
        // TODO: Implement toArray() method.
    }
```

4.2 Signature

Task Signatures

Signatures may be created by calling the `\\Rhubarb\\Rhubarb::task` method. Providing the `name`, optional arguments, properties and headers will return a signature. You may use this signature in various ways in your workflow; either as means to call the task or as a template for many tasks.

```
$sig = $rhubarb->task('tasks.add', Args::newArgs(Python::LANG, 2, 2), array('countdown' => 10));
// tasks.add(2, 2)

$add = $rhubarb->t('task.add');
$add->s(Args::newArgs(Python::LANG, 2, 2));
// tasks.add(2, 2)
```

This example demonstrates how you may access the properties, args and headers defined within the signature.

```
$add = $rhubarb->task('task.add', Args::newArgs(Python::LANG, 2, 2, Kwargs::newKwargs(array('arg1' =
// tasks.add(2, 2, arg1=1, arg2=2)
$add->getArgs();
// array(
//     'args' => array(1, 2),
//     'kwargs' => array('arg1' => 1, 'arg2' => 2)
// );
$add->getHeaders();
// array(
//     'lang' => 'py',
//     'c_type' => 'tasks.add'
// );
```

Executing the signature may be down in two ways `delay` or `applyAsync`. `delay` is a wrapper of the other to provide familiarity with the Celery API.

```
$add = $rhubarb->t('task.add');
$result = $add->applyAsync(Args::newArgs(Python::LANG, 2, 2));
$result->get();
// 4

$add = $rhubarb->t('task.add');
$result = $add->delay(Args::newArgs(Python::LANG, 2, 2), array(), array('countdown' => 1 ));
$result->get();
// 4
```

4.3 Chains

Task Chains

Creating a task chain Example 1

```
use Rhubarb\Rhubarb;
use Rhubarb\Task\Args\Python;

$config = include('configuration/predis.php');
$rhubarb = new Rhubarb($config);

$sig = $rhubarb->task('app.add');

$chain = $rhubarb->chain();
for ($i = 0; $i < 10; $i++) {
    $chain->push($sig->s(new Python($i, $i * 10)));
}
$asyncResult = $chain();
```

Creating a task chain Example 2

```
use Rhubarb\Rhubarb;
use Rhubarb\Task\Args\Python;

$config = include('configuration/predis.php');
$rhubarb = new Rhubarb($config);
$sig = $rhubarb->task('app.add');

$tasks = array();
for ($i = 0; $i < 10; $i++) {
    $tasks[] = $sig->s(new Python($i, $i * 10));
}
$chain = $rhubarb->chain($tasks);
$asyncResult = $chain();
```

4.4 Groups

Task Groups

TBD

4.5 Chords

Task Chords

TBD

CHAPTER 5

Indices and tables

- *genindex*
- *modindex*
- *search*