RabbitMQ with PHP

* You Can Use RabbitMQ with PHP using the following package [php-amqplib/php-amqplib](https://github.com/php-amqplib/php-amqplib)
* You can install this package via composer using this command:

**composer require php-amqplib/php-amqplib**

* Create configuration file for handling RabbitMQ connection setting for ex: **config.php** and set your connection configuration in ex:

<?php

define('HOST', 'localhost');

define('PORT', 5672);

define('USER', 'guest');

define('PASS', 'guest');

define('VHOST', '/');

* *Create your publisher*

<?php

require(\_\_DIR\_\_.'/vendor/autoload.php');

include(\_\_DIR\_\_ . '/config.php');

use PhpAmqpLib\Connection\AMQPStreamConnection;

use PhpAmqpLib\Exchange\AMQPExchangeType;

use PhpAmqpLib\Message\AMQPMessage;

$exchange = 'subscribers';

$queue = 'subscribers\_accounts';

$connection = new AMQPStreamConnection(HOST, PORT, USER, PASS, VHOST);

$channel = $connection->channel();

*/\**

*The following code is the same both in the consumer and the producer.*

*In this way we are sure we always have a queue to consume from and an*

*exchange where to publish messages.*

*\*/*

*/\**

*name: $queue*

*passive: false //if the queue is already exist return an error*

*durable: true // the queue will survive server restarts(Metadata of a durable queue is stored on disk, Durable queues will be recovered on node boot, including messages in them published as persistent)*

*exclusive: false // the queue can be accessed in other channels*

*auto\_delete: false //the queue won't be deleted once the channel is closed.*

*\*/*

$channel->queue\_declare($queue, false, true, false, false);

*/\**

*name: $exchange*

*type: direct*

*passive: false //if the queue is already exist return an error*

*durable: true // the exchange will survive server restarts(Metadata of a durable queue is stored on disk, Durable queues will be recovered on node boot, including messages in them published as persistent)*

*auto\_delete: false //the exchange won't be deleted once the channel is closed.*

*\*/*

$channel->exchange\_declare($exchange, AMQPExchangeType::DIRECT, false, true, false);

$channel->queue\_bind($queue, $exchange);

$faker = Faker\Factory::create();

for ($i = 0; $i < 100000; $i++) {

    $messageBody = json\_encode(['name'=>$faker->name, 'email'=>$faker->email, 'phone'=>$faker->PhoneNumber(),  'address'=>$faker->address]);

    $message = new AMQPMessage($messageBody, ['content\_type' => 'application/json', 'delivery\_mode' => AMQPMessage::DELIVERY\_MODE\_PERSISTENT]);

    $channel->basic\_publish($message, $exchange);

}

echo 'Finish publishing to queue'.*PHP\_EOL*;

$channel->close();

$connection->close();

* Create consumer

<?php

require(\_\_DIR\_\_.'/vendor/autoload.php');

include(\_\_DIR\_\_ . '/config.php');

use PhpAmqpLib\Connection\AMQPStreamConnection;

use PhpAmqpLib\Exchange\AMQPExchangeType;

$exchange = 'subscribers';

$queue = 'subscribers\_accounts';

$consumerTag = 'consumer';

$connection = new AMQPStreamConnection(HOST, PORT, USER, PASS, VHOST);

$channel = $connection->channel();

*/\**

*The following code is the same both in the consumer and the producer.*

*In this way we are sure we always have a queue to consume from and an*

*exchange where to publish messages.*

*\*/*

*/\**

*name: $queue*

*passive: false //if the queue is already exist return an error*

*durable: true // the queue will survive server restarts(Metadata of a durable queue is stored on disk, Durable queues will be recovered on node boot, including messages in them published as persistent)*

*exclusive: false // the queue can be accessed in other channels*

*auto\_delete: false //the queue won't be deleted once the channel is closed.*

*\*/*

$channel->queue\_declare($queue, false, true, false, false);

*/\**

*name: $exchange*

*type: direct*

*passive: false //if the queue is already exist return an error*

*durable: true // the exchange will survive server restarts(Metadata of a durable queue is stored on disk, Durable queues will be recovered on node boot, including messages in them published as persistent)*

*auto\_delete: false //the exchange won't be deleted once the channel is closed.*

*\*/*

$channel->exchange\_declare($exchange, AMQPExchangeType::DIRECT, false, true, false);

$channel->queue\_bind($queue, $exchange);

*/\*\**

*\* @param \PhpAmqpLib\Message\AMQPMessage $message*

*\*/*

function process\_message($message)

{

    $message\_body = json\_decode($message->body);

    writeDataToFile($message\_body);

    $message->ack();

*// Send a message with the string "quit" to cancel the consumer.*

    if ($message->body === 'quit') {

        $message->getChannel()->basic\_cancel($message->getConsumerTag());

    }

}

*/\**

*queue: Queue from where to get the messages*

*consumer\_tag: Consumer identifier*

*no\_local: Don't receive messages published by this consumer.*

*no\_ack: If set to true, automatic acknowledgement mode will be used by this consumer. See https://www.rabbitmq.com/confirms.html for details.*

*exclusive: Request exclusive consumer access, meaning only this consumer can access the queue*

*nowait:*

*callback: A PHP Callback*

*\*/*

$channel->basic\_consume($queue, $consumerTag, false, false, false, false, 'process\_message');

*/\*\**

*\* @param \PhpAmqpLib\Channel\AMQPChannel $channel*

*\* @param \PhpAmqpLib\Connection\AbstractConnection $connection*

*\*/*

function shutdown($channel, $connection)

{

    $channel->close();

    $connection->close();

}

register\_shutdown\_function('shutdown', $channel, $connection);

*// Loop as long as the channel has callbacks registered*

while ($channel->is\_consuming()) {

    $channel->wait();

}

function writeDataToFile($message)

{

    echo $message->name . "\n";

    $folder\_location = 'uploads/';

    $file\_path = 'uploads/' . strtolower(trim(str\_replace(' ', '-', $message->name)));

*//check path*

    if (!file\_exists($folder\_location)) {

        mkdir($folder\_location, 0777, true);

    }

*//check file to open*

    $file\_object = fopen($file\_path, 'a+');

*//write data to file*

    fwrite($file\_object, $message->name . "\n" .$message->email. "\n" .$message->phone. "\n" .$message->address);

    fclose($file\_object);

}