Async PHP

via PCNTL extension

shipmonk

@molsavsky1

Speaker

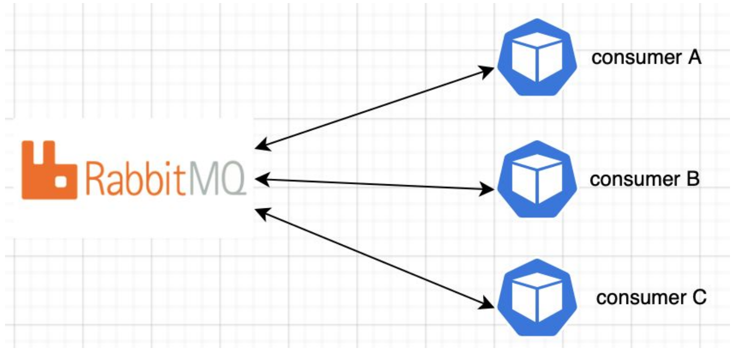
- Michael Olšavský
    - 4 years in ShipMonk
    - Mostly DX / Internals across all teams, Technical hiring
    - Social
        - github.com/olsavmic
        - @molsavsky1

What? Why?

Async PHP via PCNTL extension

Q/A: sli.do/shipmonk

Problem instance 1



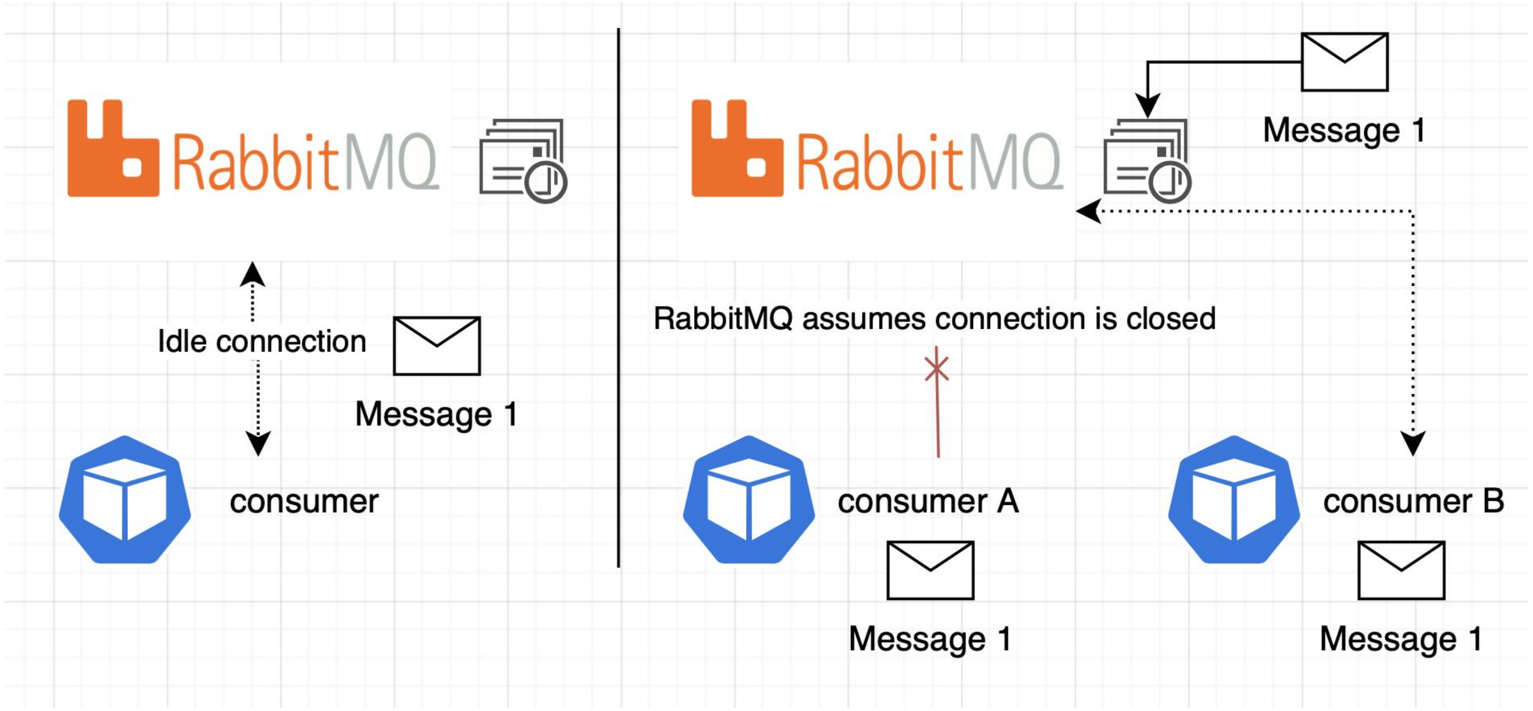
- Long-running CLI app
- Persistent connection
- Some async tasks may take minutes to complete
- But we need to keep the connection alive on both sides

Specifically for us

- Symfony application
- Consuming and producing messages via RabbitMQ (AMQP protocol)

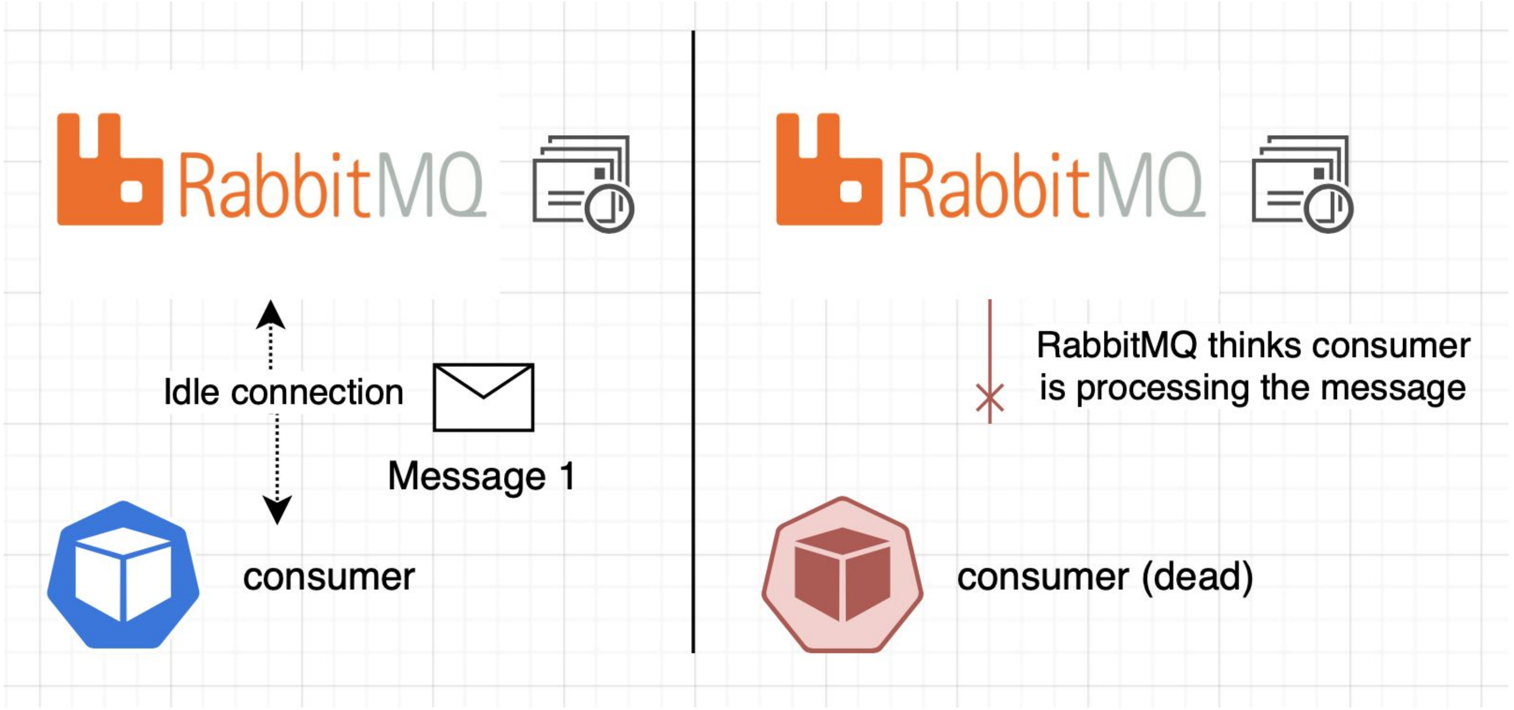
RabbitMQ Heartbeats

Too short heartbeats



Message processed twice!

Too long heartbeats



Dead time, message waiting!

Goals

1. Detect broken connections and automatically perform failover
2. Prevent termination of idle connections
3. Introduce **minimum business code changes**
4. Minimum performance overhead

Manual solution

```
foreach ($orders as $order) {
   $this->someFacade->someCallTakingTooLong($order);

   // this sends heartbeat to all RabbitMQ connections
   $this->connectionManager->pingAll();
}
```

Manual solution

```php
public function dispatchImmediately(DispatchableConsumerMessage $message): void
{
    try {
        /** @throws AMQPRuntimeException */
        $this->producer->publish($message);
    } catch (AMQPHeartbeatMissedException $e) {
        $this->logger->logInfoMessage('AMQP Reconnecting after missed heartbeat');
        $this->connection->reconnect();
        $this->producer->publish($message);
    }
}
```

❌ Manual Solution

- Spreads across the codebase
- Prevents the problem only locally
- Issues will occur until someone proactively fixes the problem

Can we do it **async?**

Async PHP

- Event-loop based solutions
  - ReactPHP
  - AMPHP
- pcntl_fork
  - spatie/async
- parallel extension (https://www.php.net/manual/en/book.parallel.php)
  - Message passing via channels
  - Requires `--enable-zts`

PCNTL extension

- Unix-like Process Control (https://www.php.net/manual/en/intro.pcntl.php)
- Supported by CLI and CGI, not FPM or mod_php
- **SIGINT**, **SIGTERM**, **SIGKILL**, ...

Support for:

- Process management
- System signal handling

`pcntl_async_signals(true)` available since PHP 7.1

Async tasks
with PCTNL Alarm

```
const INTERVAL_IN_SECONDS = 5;

pcntl_async_signals(true);

pcntl_signal(SIGALRM, static function (): void {
   pcntl_alarm(INTERVAL_IN_SECONDS);
});

pcntl_alarm(INTERVAL_IN_SECONDS);
```

```php
pcntl_async_signals(true);

pcntl_signal(SIGALRM, static function () use ($connection, $interval): void
{
    // ...
    $connection->checkHeartBeat();
    // …

    pcntl_alarm($interval);
});

pcntl_alarm($interval);
```

Restrictions

Beware

- Signal handlers are blocking the main execution
  - **Set strict timeouts** for **code** running **inside the handlers**!
- Blocking native calls (curl, PDO::exec, ...) are uninterruptible by default
  - **Set** sane **timeouts** for **all application code**!
- Interrupted `sleep($seconds)` does not resume

Interrupt-safe sleep

```php
/**
* Interrupt safe sleep
*/
public static function sleep(int $seconds): void
{
   do {
       $seconds = sleep($seconds);
   } while ($seconds !== 0);
}

/**
* Interrupt safe usleep
*/
public static function usleep(int $microseconds): void
{
   $seconds = (int) ($microseconds / self::MICROSECONDS_IN_SECOND);

   self::sleep($seconds);
   // usleep doesn't return remaining microseconds so we must use only for the sub-second part
   usleep($microseconds - $seconds * self::MICROSECONDS_IN_SECOND);
}
```

✅ PCNTL-based solution

- Almost no code modification - developers don’t need to worry
- Zero-performance cost (with `pcntl_async_signals(true)`)

Problem instance 2

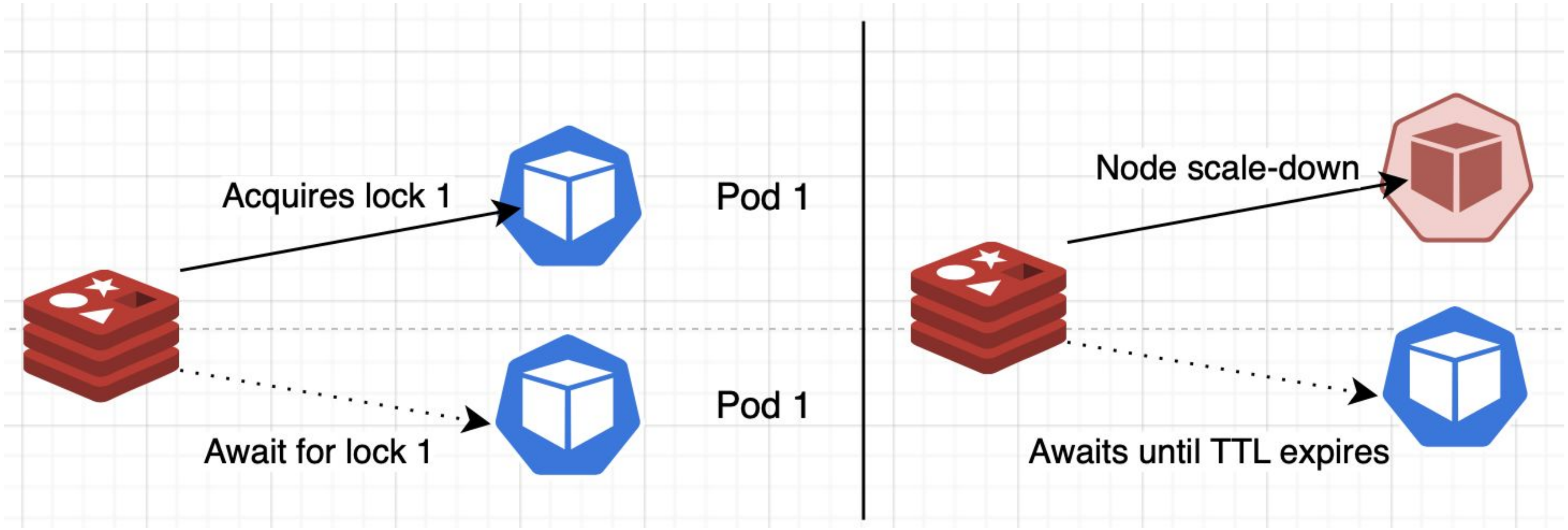
- Long-running CLI app
- Releasing acquired resources on shutdown

Specifically

- Symfony application
- Application acquires **atomic lock with TTL** via **Redis**

Deployments

Node scale-downs



Our **testing team** was **complaining the most**

Graceful shutdown
with signal handlers

What is Graceful Shutdown?

- **“Graceful shutdown is a process of shutting down an application in a way that all pending tasks are either completed or intentionally rejected.”**

- The configuration for web-server is completely different from CLI
  - Load-balancers
  - Nginx
  - PHP-FPM
  - Great article → https://tinyurl.com/graceful-shutdown-fpm

```php
$this->semaphore->runOrWait(
    SharedLockKey::createPickingJobsLockKey(),
    function () use ($message): void {
        $this->someFacade->someActionToRunUnderLock(
            // What if the app stops inside?
            $message->getPickingJobIds()
        );
    },
    maxWaitTimeInSeconds: 10,
    ttlInSeconds: 15,
);
```

How Kubernetes kills pods

- All containers in a pod receive SIGTERM first
- Default 30s period for shutdown procedures
  - Configurable `terminationGracePeriod`
  - SIGKILL after grace period
- OOM is always SIGKILL 😢

```php
pcntl_async_signals( enable: true);

pcntl_signal( signal: SIGUSR1, function () {
    throw new Exception( message: 'SIGUSR1');
});

function foo()
{
    try {
        posix_kill(posix_getpid(),  signal: SIGUSR1);
    } catch (Exception $e) {
        echo "Caught exception: {$e->getMessage()}\n";
        echo $e->getTraceAsString();
    }
}

foo();
```

Signal handler is executed on top of current stack

```
Caught exception: SIGUSR1
#0 async-signals/scripts/callstack.php(12): {closure}(30, Array)
#1 async-signals/scripts/callstack.php(19): foo()
#2 {main}
```

shipmonk

In our semaphore implementation...

```
try {
   return $callback($acquiredLocks);
} finally {
   $this->releaseAcquiredLocks($acquiredLocks);
}
```

During app initialization...

```
pcntl_async_signals(true);

pcntl_signal(SIGTERM, static function (): never {
    throw InterruptedBySignalException::sigterm();
});
```

shipmonk-rnd/pcntl-signal-manager

- Nicer API
- Object-scoped signal handlers (without memory leaks!)
- Multiple signal handlers for single signal
- Manages previously registered handlers

Soon:

- RepeatingTask API (SIGALRM)

Summary

- PCNTL signal handlers are a quick win
  - Reliable
  - No business code changes
- If it doesn’t work in your application, there are probably some other issues
  - Typically missing timeouts for blocking calls
- **Do not combine PCNTL handlers** with **event-loop based** solutions
- Signal handlers **are blocking!**
- Signal handlers run **on top of current execution stack**

Questions?

sli.do/shipmonk

Thank you!

github.com/shipmonk-rnd