Worker

Last edited by **Maelan LE BORGNE** 4 years ago

Worker

L'asynchronisme sur ePack-API est géré par la délégation de tâche par le biais de messages. Ce système s'articule autour d' :

- Un gestionnaire de message (**RabbitMQ**)
- Un gestionnaire de processus (**Supervisor**)
- Des tâches cli, qui consomment et traitent des messages (**Worker**)

Pour un exemple complet de l'implémentation d'un nouveau worker, voir le cookbook dédié

Définition

Les Workers sont des tâches constamment à l'écoute des messages en provenance de la queue qui leur est assignée. Ils savent comment traiter ces messages, les potentielles erreur et ont leur configuration propre. Il devrait toujours y avoir au moins 1 instance d'un type de worker en activité, et l'on peut théoriquement ajouter une infinité de workers, sur un même serveur ou sur des environnements distincts, pour paralléliser le travail.
Les Workers sont définies par des classes qui héritent de WorkerTask. Le mécanisme de base permettant de consommer des messages y est implémenté.

Configuration

La majorité des propriété héritées de WorkerTask peuvent être remplacée. La configuration d'un worker devrait toujours être faite dans la méthode `initialize`, et si possible après l'appel à `parent::initialize()`.

- TTL (Time-To-Live) : Durée maximum d'exécution du processus, en secondes
- queue : Queue dans laquelle consommer les messages. Utiliser la méthode `setQueue("nom_de_la_queue")`
- spare_memory_multiplier : Facteur de quantité de mémoire à réserver pour être libérée lors de l'arrêt inopiné du script. Se défini directement dans la déclarations des propriétés ou avant l'appel à `parent::initialize()`
- logger : Classe d'assistance au logs. Définir le logger avec la méthode `$logger->setMainLogger("nom_du_logger")`

Callback

Pour définir ce qu'un worker doit faire avec le message, il faut implémenter la méthode `callback(MessageAdapter $message)`
A chaque message consommé dans la queue, la fonction `callback` sera appelée, avec l'argument `message` qui est une instance de MessageAdapter construite à partir du message récupéré.

⚠️ Pensez à `acknowledge` le message à la fin de votre fonction callback ⚠️

Shutdown

PHP ayant une fâcheuse tendance à s'arrêter de manière brutale, on garde-fou a été mise en place afin d'avoir plus de contrôle sur le comportement et l'état de l'application en cas d'arrêt inopiné d'un worker.
Avant chaque exécution de la méthode `callback`, le ShutdownHelper active une fonction qui sera exécutée à l'arrêt du processus. Après l'exécution de `callback`, cette fonction est désactivée.
La fonction exécutée par ShutdownHelper en cas d'arrêt est la fonction `WorkerTask::shutdown()`. Par défaut, cette fonction génère un log CRTITCAL et *nack* le message, mais ce comportement peux être modifié en surchargeant cette fonction.
Un système de libération de mémoire est mis en place pour pouvoir exécuter la fonction `shutdown` même en cas d'arrêt à cause d'une erreur fatal de type "Allowed Memory Size Exhausted". Si le traitement de la fonction demande plus de mémoire que celle allouée par défaut, augmenter *spare_memory_multiplier*.