Hobo - Development #26961

optimisation provisionning en multi-publik / mono-rabbitmq

03 octobre 2018 22:34 - Frédéric Péters

Statut:	Nouveau	**Début:**	03 octobre 2018
Priorité:	Normal	**Echéance:**	
Assigné à:		**% réalisé:**	0%
Catégorie:		**Temps estimé:**	0:00 heure
Version cible:			
Patch proposed:	Non	**Planning:**	

Description

Sur une configuration avec tous les services sur une même machine, un seul rabbitmq, et du multi-publik, mettre un usager dans un rôle provoque :

```
{u'audience': [u'https://agendas-collectivite1/accounts/mellon/metadata/', u'https://demarches-collectivite1/saml/metadata', u'https://passerelle-collectivite1/accounts/mellon/metadata/', u'https://collectivite1/accounts/mellon/metadata/', u'https://agents-collectivite1/accounts/mellon/metadata/'], u'full': False, u'@type': u'provision', u'objects': {u'data': [{u'emails_to_members': True, u'description': u'', u'name': u'Debug XXX', u'emails': [], u'details': u'', u'slug': u'debug-...', u'uuid': u'f47e68deb0e34dcabd573e3f243e59b7'}], u'@type': u'role'}}
```

reçu pour les n applications, donc n fork() pour faire des hobo_notify, alors qu'il n'y a pas eu de changement sur le rôle.

Puis,

```
{u'audience': [u'https://hobo-collectivite3.toodego.com/accounts/mellon/metadata/'], u'full': False, u'@type': u'provision', u'objects': {u'data': [{u'last_name': ...
```

envoyé n fois, avec n = le nombre de services où l'usager a un rôle, donc à imaginer un admin chez nous, tous les services, donc hobo/combo/passerelle/wcs/bijoe/etc. * nombre de collectivités, = rapidement quelques dizaines de fork().

En comptant entre une et deux secondes par appel à hobo_notify, et une durée de vie du message rabbitmq à 120 secondes, il suffit d'un peu de traffic sur le bus et on perd des messages.

Différentes pistes donc :

- ne pas transmettre sur le bus des messages pour un rôle qui n'a pas bougé.
- améliorer la vitesse d'un hobo_notify.
- merger les messages vers différentes audiences quand le contenu est identique.
- étudier davantage rabbitmq, peut-être y a-t-il moyen de faire une queue par service, et qu'on puisse configurer pour avoir des workers par queue pour gagner une concurrence dans les traitements (par service, comme quand on les fait tourner sur différents serveurs)
- ?

Historique

#1 - 03 octobre 2018 22:46 - Frédéric Péters

- Description mis à jour

#2 - 04 octobre 2018 06:39 - Emmanuel Cazenave

..... et une durée de vie du message rabbitmq à 120 secondes ,,,,

D'où viennent ces 120 secondes ?

..... étudier davantage rabbitmq .....

Sans complètement comprendre comment ça de passe chez nous, pleins de façon jouer sur les worker et sur les queues pour faire arriver le bon message au bon destinataire dans rabbitMQ (http://www.rabbitmq.com/tutorials/tutorial-four-python.html, http://www.rabbitmq.com/tutorials/tutorial-five-python.html).

Et pensais que ces pattern n'étaient pas atteignables en utilisant Celery mais apparemment si http://docs.celeryproject.org/en/latest/userguide/workers.html#queues-adding-consumers (noter les paramètres la méthode add_consumer qui

reprennent directement les concepts de la doc rabbitmq que je pointe).

#3 - 04 octobre 2018 07:55 - Frédéric Péters

> ..... et une durée de vie du message rabbitmq à 120 secondes ,,,,
>
> D'où viennent ces 120 secondes ?

debian/debian_config_common.py:BROKER_TASK_EXPIRES = 120

#4 - 04 octobre 2018 09:57 - Benjamin Dauvergne

Je serai assez pour utiliser pika/RabbitMQ directement plutôt que celery (ne serait-ce que parce que kombu semble être un pot de pue), j'imagine quelque chose dans ce goût là:

- chaque service déclare une queue avec son nom de service (combo, hobo, wcs, etc..)
- hobo déclare un exchange avec son nom (son domaine) ce sera le point d'entrée des messages de tout le monde pour une plateforme
- hobo déclarer un exchange "publik" ce sera le point d'entré des messages de déploiement
- chaque service crée un binding pour router les messages vers son nom (son domaine) arrivant sur l'exchange du hobo principal de sa plateforme Publik

Au lieu d'avoir un hobo-agent unique avec un worker, chaque service a son propre worker, lancé par l'unit systemd (en utilisant Pika et son eventloop la plus simple, BlockingConnection, plus besoin de forker on est toujours dans le même environnement).

Error executing the **plantuml** macro (Missing partial wiki_external_filter/_macro_image with {:locale=>[:fr, :en], :formats=>[:pdf], :variants=>[], :handlers=>[:raw, :erb, :html, :builder, :ruby, :rsb]}. Searched in: * "/usr/share/redmine/plugins/wiki_external_filter/app/views" * "/usr/share/redmine/plugins/wiki_external_filter/app/views" * "/usr/share/redmine/plugins/redmine_tags/app/views" * "/usr/share/redmine/plugins/redmine_entrouvert/app/views" * "/usr/share/redmine/plugins/plantuml/app/views" * "/usr/share/redmine/plugins/localizable/app/views" * "/usr/share/redmine/app/views")

Un message broadcasté est posé sur l'exchange hobo.publik.org, avec comme clé de routage "broadcast".
Un message de déploiement est posé sur l'exchange hobo.publik.org comme clé de routage le type du service.
Un message de provisionning pour un service est posé sur l'exchange hobo.publik.org avec comme clé de routage le domaine du service.

#5 - 04 octobre 2018 09:59 - Benjamin Dauvergne

Benjamin Dauvergne a écrit :

> ...

Aussi:

- toutes les queues et les messages sont déclarés durables avec acknowledgement (on ne perd plus rien)
- à terme on peut imaginer d'utiliser ça pour communiquer entre les workflows et passerelle et ne plus se prendre le choux avec HTTP et du rejeu, passerelle obtient une queue de message pour ses connecteurs