sujet de TP RabbitMQ

Yvon Kermarrec

octobre 2016

1 - Objectifs de ce TP

- expérimenter les fonctionnalités et services du MOM (Message Oriented Middleware) RabbitMQ
- présenter des exemples des différentes configurations des exchanges, des filtres et du routage et aborder la richesse de ces configurations.

2 - Préliminaire

Nous allons utiliser la VM Unbuntu14_RabbitMQ qui est configurée pour lancer dès le démarrage un serveur RabbitMQ. Cette activation du serveur requiert des privilèges d'administrateur système et nous allons aborder quelques unes de ces commandes à la fin de ce TP

Pour cette VM nous avons 2 utilisateurs avec les id et mot de passe suivants:

- utilisateur de base (sans privilège) : user / usr
- utilisateur et administrateur: root / adm

A noter que le serveur RabbitMQ fonctionne sur votre machine locale. Pour s'y connecter, il y a donc utilisation de "localhost" qui pourra être remplacé par le nom de la machine hôte sinon.

Si vous le souhaitez, vous pouvez configurer un plugin du serveur RabbitMQ afin de visualiser les échanges de messages et les files d'attente. Pour cela, voici les commandes à exécuter (merci à Florencia Alvarez - étudiante en 2016 pour le code):

```
sudo rabbitmq-plugins enable rabbitmq_management
sudo rabbitmqctl stop
sudo invoke-rc.d rabbitmq-server start
```

lancez un navigateur web avec `http://localhost:15672`

puis précisez un compte utilisateur par défaut de RabbitMQ :

```
user: guest
pass: guest
```

3 - Texte du TP

Nous allons reprendre le tutoriel proposé par RabbitMQ à l'adresse suivante:

https://www.rabbitmq.com/getstarted.html

Python se révèle comme très adapté pour ce TP mais vous pouvez utiliser un autre langage si vous le jugez opportun!

les sources des exemples sont déposés sur :

http://formations.telecom-bretagne.eu/syst/TPDIST/TP-RabbitMQ/

3.1 Premier exemple : "Hello world"

Ce premier exemple présente les commandes de base : se connecter au serveur RabbitMQ et envoyer / recevoir un message.

Regarder en particulier le code du récepteur et notez que l'on ne programme pas l'attente d'un message mais on précise la procédure appelée via le mécanisme de 'call back'. Pour s'en rendre compte, activez plusieurs fois le processus émetteur.

Nous allons maintenant voir si RabbitMQ est sensible à l'ordre des appels. Pour cela, lancez:

- le récepteur puis l'émetteur
- l'émetteur puis le récepteur

Que se passe-t-il ? comment peut on expliquer cette situation ?

3.1 Second exemple : "work queue"

Ce second exemple montre comme une file d'attente peut 'servir' plusieurs consommateurs. Remarquez en particulier la politique du 'round robin' lorsque plusieurs programmes déposent des données dans la file d'attente.

Avec cet exemple, nous allons aussi voir comment RabbitMQ se comporte lorsqu'un processus client qui consomme un message se plante. Pour identifier cela, vous pourrez programmer une temporisation (avec un certain nombre de "." dans la chaine en entrée pour simuler une attente en lien avec l'exécution du traitement du message du coté du receveur) et puis arrêter le programme avec un CTL-C. Que se passe-t-il ? est ce que la donnée qu'il traitait est redirigée vers un autre worker?

vous pouvez aussi voir ce qui se passe si vous activez 2 récepteurs (worker.py) et 2 émetteurs (new_task.py) et que vous arrêtez un récepteur alors qu'il est en plein traitement ?

Faites le point sur les opérations en lien avec la persistance des messages (dans la file d'attente, mais aussi dans le serveur.

Regarder l'impact de l'utilisation du '*prefetch'* et comment il modifie la politique du tourniquet (ou *'round robin*' ? pour cela vous allez activer 2 récepteurs et plusieurs émetteurs : certains demandent des travaux courts (peu de points dans l'argument de la requête), d'autres simulant une demande plus longue (avec plus de "." dans l'argument de la requête).

3.3 Troisième exemple : "publish subscribe"

Dans cet exemple, nous allons voir comment on peut programmer et configurer un 'exchange'. Un exchange est utilisé par celui qui dépose un message dans RabbitMQ et à partir de là le message est dirigé vers une ou plusieurs files d'attente.

Comme nous avons un nombre quelconque de clients (et consommateurs de données depuis la file d'attente), nous allons créer une file d'attente temporaire. Regardez en particulier dans le code comment la durée de vie de la file d'attente est fixée avec l'attribut "*exclusive*".

Lors du lancement des programmes, vous pourrez identifier les files d'attente créées avec la commande :

```
sudo rabbitmqctl list_bindings
```

3.4 Quatrième exemple : "routing"

Dans cet exemple, nous allons affiner la programmation d'un exchange et voir comment des files d'attentes peuvent y être reliées. Nous allons ensuite produire des messages dans l'exchange qui vont être routés en fonction de la valeur de la clé de routage (*'routing key'*).
Pour mettre en évidence ce routage, lancez par exemple 2 récepteurs avec des attentes de type de messages différents. Vous verrez en particulier que le même message sera propagé vers les récepteurs qui ont spécifié le message attendu.

3.5 Cinquième exemple : "topics"

Dans cet exemple, nous allons voir comment les messages peuvent être redirigés de façon flexible vers les files d'attente. Nous allons voir aussi comment un client peut récupérer tous les messages émis depuis un exchange avec l'utilisation de "#" (0 ou n champs quelconques) et de "*" (une valeur quelconque pour un champ donné).

4 - Quelques commandes d'administration

toutes ces commandes permettent de gérer et de configurer le MOM. Ces instructions doivent être appelées avec le privilège de '*root*'. Vous trouverez la documentation sur :

https://www.rabbitmq.com/man/rabbitmqctl.1.man.html

- rabbitmqctl status : pour déterminer quelles sont les caractéristiques du serveur.
- rabbitmqctl add_user cashing-tier cashMe1 : pour rajouter un utilisateur
- rabbitmqctl delete_user cashing-tier : pour retirer un utilisateur
- rabbitmqctl list_users : pour récuperer la liste des utilisateurs connus
- rabbitmqctl list_queues : pour lister les files d'attente