POLYTECH NICE-SOPHIA
Département Sciences Informatiques

Université Nice Sophia Antipolis

Mini projet applications réparties SI4 : Twitter

Maylanie Mesnier [mesnier@polytech.unice.fr]
Camille Boinaud [boinaud@polytech.unice.fr]

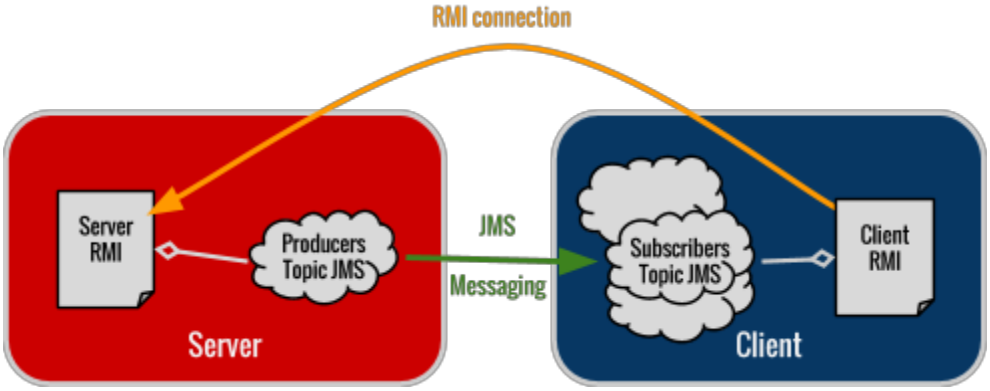
Ce projet a pour but de modéliser un mini réseau social tel que Twitter. De manière très simple, on publie un message rattaché à un thème (hashtag), et tous les utilisateurs abonnés à ce hashtag pourront ainsi le recevoir. Notre application a dû respecter deux contraintes : la gestion des actions de l'utilisateur est effectuée par un serveur RMI et les messages sont publiés à travers des topics JMS (grâce au MOM ActiveMQ).

Fonctionnalités du projet

Les fonctionnalités que nous avons choisies de développer sont :

- ❏ gestion du compte utilisateur (avec la création d'un compte, la connexion et la déconnexion)
- ❏ publication d'un message (avec création d'un nouvel hashtag s'il n'existait pas encore, ou publication sur le topic JMS qui lui a déjà été associé plus tôt)
- ❏ possibilité de s'abonner à un hashtag donné, puis de s'en désabonner
- ❏ possibilité de lister l'ensemble des hashtags existants, ainsi que ceux auxquels je me suis abonné
- ❏ afficher la liste des tweets relatifs aux hashtags que je follow et qui ont été publiés depuis ma connexion (ou tout au moins depuis que je suis abonné à ce hashtag).

Architecture du projet



Serveur :

Au niveau de notre serveur RMI, nous avons donc une interface qui est accessible depuis le client. Ce serveur va donc contenir une liste d'objets *Account* afin de pouvoir vérifier les identifiants d'un utilisateur connecté, puis d'enregistrer la connexion. C'est aussi par cet objet *Account* que l'on va pouvoir enregistrer les hashtags suivis par un utilisateur donné.

Le serveur va aussi contenir le producteurs de messages JMS. Ainsi, le serveur va pouvoir envoyer les messages concernant un hashtag précis en appelant la méthode *publish()* de la classe *PubJMS*. Les messages pourront ainsi transiter à travers les topics JMS pour être diffusés auprès des tous les abonnés de son hashtag coté client (grâce au MOM ActiveMQ).

Client :

Au niveau du client, on va utiliser les différentes méthodes de notre serveur pour contrôler (ou initialiser) l'identité de nos utilisateurs au niveau de l'application.

Le client possède une liste de *SubJMS* afin d'avoir une instance du consommateur JMS pour chacun des hashtags suivis par l'utilisateur. Ces consommateurs, suite à leur instanciation restent en écoute des évènements qui se produisent sur le topic JMS auquel ils sont connectés. Ainsi, dès qu'un nouveau message arrive, il est placé dans une collection au niveau du client, l'utilisateur peut ainsi les afficher selon ses envies.

Difficultés rencontrées

Nous avons pu observer quelques difficultés pour comprendre d'où venaient certaines erreurs dues à ActiveMQ. Nous nous sommes en fin de compte rendu-compte, grâce à votre aide, qu'il s'agissait d'une erreur de nommage de nos topics au niveau du producteur et du consommateur JMS.

Aide à compilation et exécution

- ❏ Pour lancer ActiveMQ et compiler le projet, exécuter le script : ***compile.sh***
- ❏ Pour lancer l'éxecution du serveur, ouvrez un nouveau terminal et exécuter le script : ***server.sh***
- ❏ Pour lancer différentes instances du client, ouvrez un nouveau terminal (un par instance de client) et exécuter le script : ***client.sh***

OS et Versions :

- ❏ Cette description été testée sous un environnement Ubuntu 14.04 LTS 64 bits.
- ❏ La version de ActiveMQ (livrée dans le projet) est la *5.9.0.*
- ❏ Le projet est basé sur la version 1.7 de Java.