NoSQL

EPSI Bordeaux - 2018
Benjamin Chenebault
(contact@benjaminchenebault.net)

TP n°2

Concepts avancés et programmation

L'objectif de ce second TP est de prendre en main quelques concepts avancés de Redis tels que le pipelining et le publish-subscribe, appréhender l'administration d'un système Redis et développer une application qui utilise la base de données clé/valeur comme système de stockage.

EPSI

Concepts avancés

Redis est écrit en C et le projet compte environ 20 000 lignes de code, ce qui est très peu pour un projet de cette renommée. En dépit de sa taille, Redis propose une interface simplifiée qui accepte des valeurs en texte clair (plain text) sur le protocole TCP. Le protocole conçu par Redis porte le nom de **RESP** pour **REdis Serialization Protocol**.

Le protocole RESP présente les avantages suivants :

- Simple à implémenter
- Rapide à parser
- Compréhensible par un être humain

De fait, il est possible d'interagir avec la base de données sans utiliser notre client *redis-cli.* Simplement en 'streamant' les commandes via une connexion TCP.

Ouvrez une connexion TCP vers votre Redis local en utilisant la commande *telnet*.

Conseil : Les pages du man peuvent vous aider. Le port de REDIS est le 6379.

En cas de succès, vous devriez voir les messages suivants.

Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

Puis tapez les commandes suivantes, séparées par un saut de ligne

- **SET test hello**
- **SET test hello hello**
- **GET test**
- **SADD stest 1 99**
- **SMEMBERS stest**

Quelles sont les messages retournés par le serveur ?

A quoi correspondent les préfixes "+", "-", ":" des messages du serveur ?

Pipelining

Il est également possible de grouper plusieurs commandes dans une même trame TCP. Cette astuce permet d'éviter les N aller-retours client-serveur, nécessaires à l'exécution de N commandes. Le réseau peut parfois souffrir de lenteur, et votre connexion TCP peut traverser un ou plusieurs serveurs intermédiaires, ce qui peut réduire considérablement le temps entre le moment où la commande est

émise par le client et le moment où le client reçoit une réponse. Dans le jargon client-serveur, le temps d'aller-retour est appelé *round-trip time*.

Dans un terminal Unix, exécutez la commande suivante :

echo "ECHO hello" | nc localhost 6379

Puis la commande suivante :

echo -en "PING\r\nPING\r\nPING\r\n" | nc localhost 6379

Dans ces deux exemples, un seul message TCP est émis sur le réseau, ce qui peut accroître considérablement le temps de réponse si les machines ne sont pas sur le même réseau (ex : vous possédez serveur applicatif sur un serveur chez OVH et vous utilisez les services de Redis Cloud).

A quoi sert la commande *nc* ?

A quoi correspond le \r\n ? Pourquoi est-il nécessaire après chaque commande ?

Quel est le rôle des paramètres *-e* et *-n* de la commande echo ?

Publish-Subscribe

Dans le TP précédent, nous avons mis en place une file d'attente bloquante en utilisant une liste et la commande *BPOP.* Les données étaient insérées dans une liste qui se comportait comme une queue et étaient récupérées par un consommateur en attente. Un nombre illimité de messagers pouvaient émettre vers cette queue et un unique consommateur pouvait les lire, sans limite.

Ce mécanisme est puissant mais limité car le consommateur est unique. Dans d'autres circonstances, nous aimerions disposer du comportement inverse, où un ensemble d'abonnées souhaitent récupérer une information transmise par un unique émetteur.

Pour cela, Redis fournit un mécanisme de **publish-subscribe** (également appelé pub-sub).

Améliorons donc notre système de commentaires du dernier TP.

Démarrez deux session en écoute sur une clé Redis, que l'on appelle *channel* dans la nomenclature du pub-sub, avec la commande SUBSCRIBE.

redis 127.0.0.1:6379> SUBSCRIBE comments

Utilisez une troisième session pour jouer le rôle du client. Trouvez la commande pour émettre sur le channel *comments.* Vérifier que les deux sessions en écoute reçoivent toutes les deux les messages émis.

Quel est le rôle de la commande PSUBSCRIBE ?

Administration d'un serveur Redis

Le fichier de configuration Redis

Avant de modifier la configuration du serveur Redis, jeton sun oeil aux paramètres actuels de votre serveur avec la commande INFO. La commande INFO retourne un ensemble d'informations sur le serveur, la version, le PID, la mémoire utilisée, l'uptime, etc…

Cette commande est très utile car elle donne un aperçu de l'état d'un serveur en cours de fonctionnement. Elle fournit également des informations sur la fragmentation de la mémoire et le statut des réplicas. Le succès de Redis s'explique également par sa richesse en terme de configuration.

Utilisez la commande INFO et jetez un oeil aux champs retournés.

Redis est installé par défaut avec un fichier de configuration situé dans le répertoire */etc/redis*. Les commentaires du fichier sont suffisamment clair et explicit pour comprendre le sens de la plupart des paramètres.

Trouvez le fichier sur votre disque, faites en une sauvegarde et ouvrez le.

1. **Modifiez l'emplacement de votre fichier de log, et redémarrer votre serveur. Vérifiez que la modification a bien été prise en compte.**

2. **Activez l'authentification par mot de passe et vérifiez que votre serveur est bien protégé. Pour cela, connectez vous à votre base et exécutez une commande. Si vous n'êtes pas authentifié, cette action ne sera pas possible.**

Note : Lorsqu'on modifie un fichier de configuration de service, il est toujours conseillé de commencer par une sauvegarde config.bak. De cette manière, si vos modifications entraînent un arrêt du service, vous pourrez rapidement remettre en état votre serveur en reprenant le fichier sauvé.

La persistance sur disque

Redis propose différentes options de sauvegarde. La première consiste à tout garder en mémoire simplement. Si vous utilisez Redis comme un cache basique, c'est un choix raisonnable puisque la persistance augmente la latence du serveur. La seconde consiste à configurer Redis pour qu'il effectue des sauvegardes de la mémoire sur disque de façon asynchrone et transparente pour le client.

La commande LASTSAVE vous affiche la date de la dernière sauvegarde en timestamp.

Exécutez la commande sur votre poste ? De quand date la dernière sauvegarde ?

Note : La valeur retournée est un timestamp UNIX, trouvez une commande shell pour convertir ce timestamp en date.

Il est possible de forcer la sauvegarde de la base avec la commande SAVE.

Si vous jetez un oeil aux logs du serveur, vous devriez voir des messages ressemblant aux lignes ci-dessous.

[46421] 10 Oct 19:11:50 * Background saving started by pid 52123
[52123] 10 Oct 19:11:50 * DB saved on disk
[46421] 10 Oct 19:11:50 * Background saving terminated with success

Il est possible de paramétrer relativement finement le déclenchement des sauvegardes, que l'on appelle *snapshot* (action de snapshotting), à l'aide de la propriété *save* du fichier de configuration.

Lisez en détail la section du fichier redis.conf relatif au paramètre *save.*

En considérant la configuration suivante sur la base de données :

save 60 500

Que se passe t-il lorsque 3 opérations d'écriture ont lieu sur un intervalle proche ?
Que se passe t-il lorsque 900 opérations d'écriture ont lieu sur un intervalle proche ?

Puis, avec la commande suivante

save 600 1

Que se passe t-il lorsque 3 opérations d'écriture ont lieu sur un intervalle proche ?
Que se passe t-il lorsque 900 opérations d'écriture ont lieu sur un intervalle proche ?

La réplication master/slave

Comme la plupart des systèmes NoSQL, Redis supporte la réplication master-slave ou maître-esclave en français. Un serveur est maître par défaut s'il n'est définit comme l'esclave d'aucun autre. Les données sont automatiquement répliquée sur l'ensemble des esclaves.

En production, une réplication master-slave est indispensable afin d'assurer :

- la disponibilité, si le nombre d'utilisateur augmente
- le failover, si votre serveur principal tombe, un second prend le relais

Mettre en place un serveur esclave est très simple avec Redis.

Dans un premier temps, effectuez une copie de votre fichier de configuration.

$ cp redis.conf redis-s1.conf

Le fichier va rester quasi-identique à la différence des paramètres ci-dessous qu'il faut définir de la manière suivante.

port 6380 # ou un autre port de votre machine
slaveof 127.0.0.1 6379

Puis démarrer votre serveur esclave avec la commande suivante,

$ redis-server redis-s1.conf

Connectez sur votre serveur maître et enregistrez un SET avec un contenu quelconque.

127.0.0.1:6379> SADD myset 1 2 3

Enfin, vérifiez que le SET est bien présent sur le serveur esclave.

127.0.0.1:6380> SMEMBERS myset

Ajoutez un élément dans le SET depuis le serveur esclave. Qu'observez vous ?

TP DE PROGRAMMATION

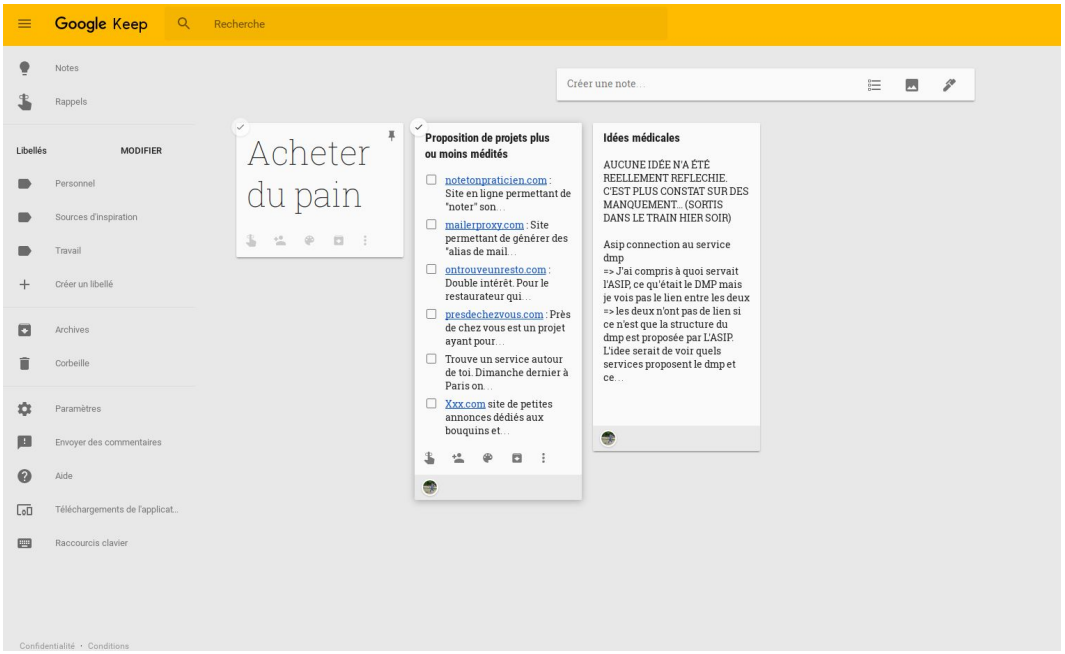
1. **A l'aide de votre langage de programmation favori (java, golang, javascript ou python) et d'un driver Redis, développez un programme permettant**
    a. **d'insérer un nombre en base**
    b. **de l'incrémenter**

    Ces deux opérations devront être effectuée au sein d'une transaction Redis.

    Puis insérez dans un set Redis 1 000 000 nombre allant de 1 à 1 000 000 et affichez (sur la sortie standard) le temps total d'insertion.

2. **Développez un second programme capable de lire des commentaires issus d'une liste bloquante Redis nommée "*comments*". Votre programme sauvera les commentaires sur le fichier /tmp/urls.log dans votre machine.**

3. **Connaissez vous Google Keep ou Evernote ? Ce sont des applications en ligne de prise de note comme celle présentée ci-dessous.**



Vous allez développer une application capable de sauvegarder et de consulter des notes. Bien entendu il n'est pas demander de développer une interface, uniquement une application serveur.

Libre à vous de choisir le type de données Redis que vous utiliserez.

La création d'une note s'effectuera par l'envoi d'une requête HTTP POST sur votre application. Par exemple, la commande suivante permettra de créer une note.

```
curl -H "Content-Type:text/plain" --data 'Penser au pain' http://localhost:8080/notes
```

La récupération/lecture des notes s'effectuera par une requête HTTP GET sur votre application.

La commande suivante vous permettra donc de récupérer la liste des toutes les notes.

```
curl http://localhost:8080/notes
```

BONUS :

- **Implémenter le path HTTP qui permet de supprimer une note.**
  (`curl -XDELETE http://localhost:8080/notes`/{id_note})

- **Enrichissez votre application, pour qu'une note porte également l'information d"auteur" et de "date de création"**

- **Mettez en place un mécanisme d'abonnement à un auteur. L'abonnement à un auteur permet de recevoir l'ensemble des notes émises par un auteur.**
  L'abonnement s'effectuera sur le path http://localhost:8080/subscribe/{auteur}.

L'implémentation pourra être réalisée au choix en HTTP keep-alive/long polling (chunk encoding ou xstream) ou par web-socket.

Rédigez un README qui détaille le fonctionnement de vos applications, puis construisez une archive tar qui contiendra les sources de vos trois programmes et votre README.

Votre archive portera le nom suivant : ***prenom.nom{.prenom2.nom2}.tar.gz***

Ex :

- *jose.garcia.tar.gz (une personne)*
- *emilie.dupont.jacqueline.dumet.tar.gz (binôme)*

Envoyez la à l'adresse benjamin.chenebault@epsi.fr avant le 20 mars 2018.