TP 5.6 REDIS - Scaling horizontal

Objectifs

Comprendre les mécanismes avancés de REDIS et notamment les outils de scaling horizontal

Exercice 1 : La persistance

Redis travaille principalement en mémoire vive. Par défaut, REDIS est en mode `RDB` et sauvegarde ses données dans un fichier `dump.rdb` selon les attributs de sauvegarde.

1. Assurez vous qu'un serveur `redis` est bien disponible
   `docker run --name monredis -v mes_data_redis:/data -p 6379:6379 -d redis:latest`
2. Ouvez deux consoles côte à côte
3. Dans la première console, relancez votre serveur Redis `monredis` puis lancez le client `redis-cli`

   Rappel : parmi toutes les commandes possibles vous pouvez
   - *vérifier le nombre de clés présentes :* `INFO keyspace`
   - *Lister les clés présentes :* `KEYS *`
   - *Détruire toutes les clés de la base courante* `flushdb`
   - *Détruire toutes les clés de toutes les bases* `flushall`
4. Dans la seconde console, connectez vous en bash sur la machine `monredis`
5. Vérifiez les paramètres de sauvegarde. Au bout de combien de temps une clé sera sauvegardée ?
6. Crééz une nouvelle clé `cpt` à la valeur 5
7. Vérifiez la date (ou la présence) de `dump.rdb` Vous constatez qu'aucune sauvegarde n'a été effectuée.
   En cas de plantage, vous auriez tout perdu !
8. La commande `SAVE` permet de "forcer" une sauvegarde sur disque. Faites le.
9. Re-vérifiez la date de `dump.rdb`
10. Changez le mode de sauvegarde pour que REDIS fasse une sauvegarde toutes les 10 sec si au moins 2 clés ont été modifiées
11. Testez le bon fonctionnement, en ajoutant 2 clés et en vérifiant à nouveau la date du fichier.
12. On passe maintenant en mode `AOF`
13. Modifiez une clé
14. Vérifiez la date des fichiers
    Cette fois REDIS sauvegarde chaque action : c'est plus sûr, mais très coûteux !

Remarque: il est possible de quitter un serveur Redis avec la commande `SHUTDOWN` qui possède deux options `SAVE/NOSAVE`. L'option par défaut est `SAVE`.

Exercice 2 : Moniteurs

Le monitoring permet, à chaud, de surveiller ce qui est envoyé sur le serveur par les différents clients. Le monitoring est une option du client Redis. Un moniteur peut être lancé n'importe quand, et pour n'importe quelle durée, juste pour voir si tout se passe bien.

1. Assurez vous d'avoir 2 consoles côte à côte
2. Dans la première console, lancez un client classique `redis-cli`
3. Dans la seconde, lancez un moniteur : `redis-cli monitor` Tant que le monitoring fonctionne le client n'est plus utilisable en interactif
4. Affectez et relisez des clés quelconques dans le premier client et observez le moniteur
5. Le client `redis-cli` possède différentes options interessantes
   `redis-cli -r 10 -i 1 <cmd>`
   l'option `-r` indique le nombre de répétitions de la commande et `-i` le délai entre 2 commandes.
   Lancez 20 fois la commande qui incrémente la clé compteur avec un délais de 1 sec
6. Le moniteur peut être démarré et arrêté quand on le souhaite. Il peut d'ailleurs y avoir plusieurs moniteurs actifs au même moment (sur différentes machines par exemple).

Exercice 3 : La réplication

La réplication permet d'assurer la haute disponibilité. Redis fonctionne en mode Maitre/Replicas (slaves) avec un serveur Maitre (il ne peut y en avoir qu'un) qui autorise lectures et écritures, et autant de réplicas que l'on souhaite, uniquement en lecture.
Idéalement chaque serveur est sur une machine différente. Avec Docker il faut créer plusieurs services, nous utiliserons cette fois une stack `docker-compose`.

1. Téléchargez le fichier `tp56_exo3.zip` qui contient un répertoire avec un `docker-compose` permettant de créer un maitre et 2 réplicas
2. Regardez attentivement chacun de ces fichiers
3. Lancez la configuration avec `docker compose up`
4. Assurez vous d'avoir 2 consoles côte à côte
5. Dans la première, lancez un client Redis sur maitre; dans la seconde un client Redis sur replica1

   A ce stade vous avez donc 2 machines, à gauche un client sur `maitre`, à droite un client sur `replica1`

6. la commande `ROLE` permet de savoir à quel type de serveur vous êtes connecté. Testez dans les deux consoles.
7. la commande `GET CONFIG slaveof` permet d'avoir les infos sur le maitre surveillé
8. Sur le maître, affectez une clé
9. Sur le réplica (slave) relisez votre clé
10. Le réplica ne fonctionne qu'en lecture. Tentez d'affecter une clé
11. `Redis` est très souple ! comme pour ls moniteurs, vous pouvez arrêter et redémarrer un réplica n'importe quand; au lancement il se resynchronise immédiatement. Arrêtez le réplica (`docker stop replica1`), affectez quelques clés sur le Maitre puis relancez le réplica. A priori toutes les clés devraient réapparaitre.
12. Une fois l'exercice terminé, arrêtez la stack `docker` avec `docker compose down -v`

Exercice 4 : Les Sentinelles

Afin d'assurer une très haute disponiblité, il est impératif de pouvoir faire face au plantage du serveur Maitre. Le rôle des sentinelles est de surveiller le Maitre et d'en élire un nouveau en cas de problème. Afin d'avoir le quorum au système de vote, il faut un nombre impair de sentinelles, dont au minimum 3 sentinelles.
La configuration minimum devient donc la suivante : 1 maitre, 2 replicas (dont l'un sera élu en cas de plantage du maitre)

et 3 sentinelles. Par ailleurs, les sentinelles necessitent impérativement un fichier de configuration.
La configuration étant assez complexe, nous mettons en place un `docker-compose` pour cela. Il s'appuie sur un Dockerfile pour configurer les sentinelles.

1. Récupérer le répertoire `tp56_exo4.zip` qui contient les fichiers nécessaires
2. Regardez attentivement chacun de ces fichiers
3. Lancez la configuration avec `docker-compose up`
   A ce stade vous avez donc 6 machines

4. Connectez vous sur `maitre` et affectez une clé `maitre="ok"`.
5. Connectez vous sur `replica1`

   (a) vérifiez qu'il est bien en mode `slave`

   (b) vérifiez aussi que la clé pécédente a bien été répliquée

   (c) Vous pouvez tenter d'affecter une clé, ça n'est évidemment pas possible.
6. Arrêtez maintenant le `maitre`
7. Attendre 10 secondes, puis reconnectez vous sur `replica1`
   (a) vérifiez qu'il est maintenant passé en mode `master`
   (b) vérifiez que la clé pécédente a bien été répliquée
   (c) affectez une nouvelle clé `replica1="ok"`. cette fois c'est possible
8. On peut pour finir vérifier que `réplica2` (pour qui rien n'a changé) connait bien toutes les clés, qu'elles aient été créées sur `maitre` comme sur `replica1`
9. Une fois l'exercice terminé, arrêtez la stack `docker` avec `docker-compose down -v`

Exercice 5 : Externaliser la gestion des sessions

Les sites web ont souvent besoin de gérer des sessions aussi bien pour l'authentification que pour le stockage de données propres à chaque utilisateur. Lors d'une montée en charge importante ($> 30.000 hits/min$, il devient alors nécessaire de dupliquer le serveurs web et distribuer les requêtes HTTP sur ce cluster de serveurs. Dans ce cas, pour éviter de perdre les données de l'utilisateur quand sa requête est envoyée sur une autre machine, il est nécessaire d'externaliser les données de la session. Ce travail est fait quasi automatiquement par le starter `spring-session-core`. Ce dernier peut utiliser n'importe quelle base de données pour stocker les sessions, et `Redis` est un candidat idéal pour cela.

Pour illustrer cela nous allons créer une application `springboot` la plus simple possible (ni `security`, ni `jsp`, ni `jpa`) juste avec `spring-session-core` et une de ses implémentations pour l'externalisation des sessions.

```
spring init --build=maven -d=web,devtools,session,jdbc,postgresql,redis -g=fr.but3 tp56
```

Q1. Version **Postgres**

1. Cette version necessite les dépendances `spring-session-jdbc` et le driver `postgresql`.
   Dans le pom généré, activez la dépendance `spring-session-jdbc` et mettez en commentaire la dépendance `spring-session-data-redis`
2. Dans le `application.properties` on indique les paramètres habituels pour créer un bean `dataSource`

   ```
   spring.datasource.url=jdbc:postgresql://psqlserv/but3
   spring.datasource.username=duchemin
   spring.datasource.password=paul
   ```

   Ainsi que ceux pour la gestion de la session via une connexion jdbc

```
spring.session.store-type=jdbc
spring.session.jdbc.initialize-schema=always
spring.session.timeout=900
```

3. On se contentera d'une simple entrée dans un controleur REST qui ajoute 2 ou 3 données quelconques en session et renvoie un "OK".

```java
package fr.but3.tp56;
import jakarta.servlet.http.HttpSession;
import org.springframework.web.bind.annotation.*;

@RestController
class MonControleur {
    @GetMapping("/")
    String home(HttpSession session) {
        session.setAttribute("dat",System.currentTimeMillis());
      return "OK!";
    }
}
```

4. Vous constatez que 2 tables sont créées : `spring_sessions` et `spring_sessions_attributes`
5. Utilisez plusieurs clients web (ou navigation privée) pour créer plusieurs sessions
6. Regardez dans Postgres le contenu des 2 tables créées pour la gestion des sessions
7. Essayez d'afficher la clé que vous avez rangé dans la session

Q2. Version **Redis**

1. Cette version nécessite les dépendances `spring-session-data-redis` avec `Redis` (A priori il est déjà dans le pom)
   Activez la dépendance `spring-session-data-redis` et mettez en commentaire la dépendance `spring-session-jdbc`
2. Mettre les paramètres propres à `Redis` (en supprimant ou commentant les 3 anciennes lignes spring.session)

   ```
   spring.session.store-type=redis
   spring.session.redis.namespace=spring:session
   spring.session.timeout=900
   ```

   Selon le `namespace` défini précédemment, les clés `Redis` sont maintenant préfixées par : `spring:session` suivi de nom de la clé
3. Assure vous que votre serveur `Redis` est accessible.
4. Eventuellement, lancez un `monitor` pour observer ce qui se passe
5. Utilisez plusieurs clients web (ou navigation privée) pour créer plusieurs sessions
6. Essayez d'afficher la clé que vous avez rangé dans la session

Deux serveurs Web peuvent maintenant partager les mêmes sessions !
On pourrait potentiellement s'authentifier sur l'un, et continuer sur l'autre