

Rapport de TP : Introduction pratique à Redis

Nom/Prénom : GNOUG Omayma

Date : 02/12/2025

Sujet : Manipulation des structures de données et administration Redis.

1. Introduction

L'objectif de ce laboratoire est de découvrir et de manipuler les principales structures et commandes de Redis, une base de données NoSQL orientée clé-valeur fonctionnant principalement en mémoire RAM (In-Memory). À travers trois étapes, nous avons exploré les types de données fondamentaux, les structures avancées (Sorted Sets, Hashes) ainsi que le système de messagerie (Pub/Sub).

2. Structures de base et commandes essentielles

Dans la première partie, nous avons pris en main les opérations CRUD sur les chaînes de caractères (Strings) et la gestion du cycle de vie des données.

```
(kali㉿kali)-[~]
$ sudo systemctl start redis-server

(kali㉿kali)-[~]
$ sudo systemctl status redis-server
● redis-server.service - Advanced key-value store
   Loaded: loaded (/usr/lib/systemd/system/redis-server.service; disabled; preset: disabled)
   Active: active (running) since Thu 2025-11-27 08:53:47 CET; 41s ago
     Docs: http://redis.io/documentation,
           man:redis-server(1)
   Main PID: 5274 (redis-server)
      Status: "Ready to accept connections"
      Tasks: 5 (limit: 2262)
     Memory: 8.2M (peak: 8.7M)
        CPU: 463ms
       CGroup: /system.slice/redis-server.service
                  └─5274 "/usr/bin/redis-server 127.0.0.1:6379"

Nov 27 08:53:47 kali systemd[1]: Starting redis-server.service - Advanced key-value store ...
Nov 27 08:53:47 kali systemd[1]: Started redis-server.service - Advanced key-value store.
```

Commandes exécutées sur les Strings :

- **SET, GET, DEL** : Création, lecture et suppression.
- **INCR, DECR** : Gestion des compteurs atomiques.

- **TTL, EXPIRE** : Définition d'une durée de vie pour le cache.

```

└$ redis-cli
127.0.0.1:6379> SET demo "Bonjour"
OK
127.0.0.1:6379> set key value [NX|XX] [GET] [EX seconds|PX milliseconds|EXAT un
127.0.0.1:6379> set key value [NX|XX] [GET] [EX seconds|PX milliseconds|EXAT un
127.0.0.1:6379> set key value [NX|XX] [GET] [EX seconds|PX milliseconds|EXAT un
127.0.0.1:6379> set key value [NX|XX] [GET] [EX seconds|PX milliseconds|EXAT un
127.0.0.1:6379> SET key value [NX|XX] [GET] [EX seconds|PX milliseconds|EXAT un
127.0.0.1:6379> SET key value [NX|XX] [GET] [EX seconds|PX milliseconds|EXAT un
127.0.0.1:6379> set key value [NX|XX] [GET] [EX seconds|PX milliseconds|EXAT un
127.0.0.1:6379> set user:1234 "omayma" [NX|XX] [GET] [EX seconds|PX millisecond
127.0.0.1:6379> SET demo "Bonjour" [NX|XX] [GET] [EX seconds|PX milliseconds|EX
127.0.0.1:6379> set user:1234 "omayma" [NX|XX] [GET] [EX seconds|PX millisecond
127.0.0.1:6379> set user:1234 "omayma"
OK
127.0.0.1:6379> get user:1234
"omayma"
127.0.0.1:6379> del user:1234
(integer) 1
127.0.0.1:6379> del user:12345
(integer) 0
127.0.0.1:6379> set 1mars 0
OK
127.0.0.1:6379> incr 1mars
(integer) 1
127.0.0.1:6379> incr 1mars
(integer) 2
127.0.0.1:6379> incr 1mars
(integer) 3
127.0.0.1:6379> decr 1mars
(integer) 2

```

Nous avons ensuite manipulé deux structures de collection :

- **Les Listes** : Ajout (**L_{PUSH/R}PUSH**), retrait (**LPOP**) et consultation (**LRANGE**).
- **Les Sets (Ensembles)** : Gestion de valeurs uniques sans ordre défini via **SADD**, **SMEMBERS** et **SREM**.

```
\$ redis> incr 1mars
127.0.0.1:6379> incr 1mars
(integer) 2
127.0.0.1:6379> incr 1mars
(integer) 3
127.0.0.1:6379> decr 1mars
(integer) 2
127.0.0.1:6379> decr 1mars
(integer) 1
127.0.0.1:6379> set macle mavaleur
OK
127.0.0.1:6379> ttl macle
(integer) -1
127.0.0.1:6379> expire macle 120
(integer) 1
127.0.0.1:6379> ttl macle
(integer) 118
127.0.0.1:6379> RPUSH mescours "Services Web"
(integer) 1
127.0.0.1:6379> RPUSH mescours "BDA"
(integer) 2
127.0.0.1:6379> LRANGE mescours 0 -1
1) "Services Web"
2) "BDA"
127.0.0.1:6379> LPOP mescours
"Services Web"
127.0.0.1:6379> LRANGE mescours 0 -1
1) "BDA"
127.0.0.1:6379> SADD utilisateurs "OMAYMA"
(integer) 1
127.0.0.1:6379> SADD utilisateurs "jamila"
```

3. Structures avancées : Sorted Sets et Hashes

Pour des besoins plus complexes comme les classements ou la structuration d'objets, nous avons utilisé des types de données spécifiques.

Ensembles Ordonnés (ZSET)

Ce type associe un score à chaque membre, permettant un tri automatique.

Commandes :

Code snippet

```
ZADD mes_scores 100 "Joueur1" // Ajout avec score
ZRANGE mes_scores 0 -1      // Affichage trié
ZRANK mes_scores "Joueur1" // Récupération de la position
```

```
__(kali㉿kali)-[~]
└─$ redis-cli
127.0.0.1:6379> ZADD score4 19 Augustin
(integer) 1
127.0.0.1:6379> ZADD score4 18 Ines
(integer) 1
127.0.0.1:6379> ZADD score4 12 Philippe
(integer) 1
127.0.0.1:6379> ZADD score4 8 Marc
(integer) 1
127.0.0.1:6379> ZRANGE score4 0 -1
1) "Marc"
2) "Philippe"
3) "Ines"
4) "Augustin"
127.0.0.1:6379> ZRANGE score4 0 4
1) "Marc"
2) "Philippe"
3) "Ines"
4) "Augustin"
127.0.0.1:6379> ZRANGE score4 0 1
1) "Marc"
2) "Philippe"
127.0.0.1:6379> ZRANK score4 Augustin
```

Hashes

Ils permettent de regrouper plusieurs champs sous une même clé, simulant un objet JSON sans schéma strict.

Commandes :

Code snippet
HSET user:1 nom "GNOUG" prenom "Omayma"
HGETALL user:1
HINCRBY user:1 age 1

```
127.0.0.1:6379> HGETALL user:4
1) "username"
2) "JAMAL"
3) "age"
4) "4"
5) "email"
6) "jamal@gmail.com"
127.0.0.1:6379> HVALS user:4
1) "JAMAL"
2) "4"
3) "jamal@gmail.com"
127.0.0.1:6379>
```

Note : Nous avons observé la différence de performance significative entre l'accès RAM (Redis) et l'accès disque.

4. Système de Publication / Souscription (Pub/Sub)

Nous avons mis en œuvre la fonctionnalité de messagerie temps réel de Redis.

- **Abonnement** : `SUBSCRIBE mescours` (Le client reste en écoute active).
- **Publication** : `PUBLISH mescours "Contenu du message"` (Diffusion instantanée aux abonnés).
- **Pattern Matching** : `PSUBSCRIBE me*` (Abonnement global à tous les canaux commençant par un motif).

```
127.0.0.1:6379> SUBSCRIBE mescours
Reading messages ... (press Ctrl-C to quit)
1) "subscribe"
2) "mescours"
3) (integer) 1
PUBLISH mescours "Un nouveau cours sur Mongo"

^C(46.57s)
127.0.0.1:6379> PUBLISH mescours "un nv cours sur Mongo"
(integer) 0
127.0.0.1:6379> PUBLISH user:4 "Bonjour"
(integer) 0
127.0.0.1:6379> PSUBSCRIBE mes*
Reading messages ... (press Ctrl-C to quit)
1) "psubscribe"
2) "mes*"
3) (integer) 1
^C(6.55s)
127.0.0.1:6379> KEYS *
1) "user:11"
2) "user:4"
3) "score4"
4) "demo"
5) "1mars"
6) "mescours"
```

5. Gestion des bases internes

Redis isole les données dans 16 bases logiques (indexées de 0 à 15). Nous avons appris à naviguer entre ces espaces.

Commandes :

Code snippet

```
SELECT 1 // Basculer vers la base n°1
KEYS * // Afficher les clés de la base active
SELECT 0 // Revenir à la base par défaut
```

```
127.0.0.1:6379> PSUBSCRIBE mes*
Reading messages ... (press Ctrl-C to quit)
1) "psubscribe"
2) "mes*"
3) (integer) 1
^C(6.55s)
127.0.0.1:6379> KEYS *
1) "user:11"
2) "user:4"
3) "score4"
4) "demo"
5) "1mars"
6) "mescours"
7) "autreutilisateurs"
8) "utilisateurs"
127.0.0.1:6379> SELECT 1
OK
127.0.0.1:6379[1]> SELECT 0
OK
127.0.0.1:6379> █
```

6-Conclusion:

Ce TP nous a permis de comprendre en profondeur les mécanismes fondamentaux de Redis et de manipuler ses principales structures de données. Grâce aux exercices réalisés, nous avons pu expérimenter la gestion de messages en temps réel via Pub/Sub, l'organisation des données au sein de plusieurs bases, et l'utilisation de commandes essentielles pour l'inspection et la modification des clés. Cette séance met en évidence la puissance et la simplicité de Redis dans les applications modernes, qu'il s'agisse de systèmes distribués, de mise en cache, de gestion d'événements ou de traitement en temps réel. Les compétences acquises au cours de ce TP constituent une base solide pour intégrer Redis dans des projets plus complexes ou orientés performance.