TP1 – Introduction à Redis

Base de Données NoSQL

NICOLAS Ethan

G5SI2
2025–2026

Table des matières

1 Introduction

Ce TP présente Redis, un système de gestion de bases de données NoSQL orienté clé/valeur. L'objectif est de comprendre son installation, ses commandes fondamentales et la manipulation de ses structures de données grâce à la CLI Redis.

Redis est intégralement en mémoire, ce qui lui confère une très haute performance. Il supporte plusieurs structures (listes, ensembles, hashmaps, sorted sets, HyperLogLog, Pub/Sub, scripts Lua, etc.).

Les manipulations présentées proviennent de trois vidéos pédagogiques :

— Vidéo 1 : Introduction à Redis

— Vidéo 2 : Ensembles ordonnés et Hashmaps

— Vidéo 3 : Pub/Sub et fonctionnalités avancées

2 Les quatre familles de bases NoSQL

Les bases NoSQL se regroupent en quatre catégories.

2.1 Bases Clé-Valeur

Stockage de paires clé/valeur. Très rapides, idéales pour le cache, les sessions ou la gestion d'état. Exemples : Redis, Memcached.

2.2 Bases Documentaires

Stockage de documents JSON/BSON. Adaptées aux données semi-structurées. Exemples : MongoDB, CouchDB.

2.3 Bases Orientées Colonnes

Optimisées pour le Big Data et l'analyse massive. Exemples : Cassandra, HBase.

2.4 Bases Orientées Graphes

Stockent des entités et leurs relations. Exemples : Neo4j, ArangoDB.
Redis appartient à la première famille mais propose des structures très riches.

3 Installation et Déploiement

3.1 Installation sous Linux

```
sudo apt update
sudo apt install redis-server
sudo systemctl enable redis
sudo systemctl start redis
redis-cli
```

3.2 Installation sous macOS (Homebrew)

```
brew update
brew install redis
brew services start redis
redis-cli
```

3.3 Installation via Docker

Installer Docker et Docker Compose

— Documentation officielle : `https://docs.docker.com/get-docker/`

3.4 Déploiement Redis avec Docker

```
docker pull redis
docker run -d --name redis-tp -p 6379:6379 redis
docker ps
docker exec -it redis-tp redis-cli
```

3.5 Déploiement avec Docker Compose

Créer un fichier `docker-compose.yml` :

```yaml
version: "3.8"

services:
  redis:
    image: redis:latest
    container_name: redis-tp
    ports:
      - "6379:6379"
    command: ["redis-server", "--appendonly", "yes"]
    volumes:
      - ./data:/data
```

Lancer :

```
docker compose up -d
docker exec -it redis-tp redis-cli
```

4 Premiers Pas avec Redis

4.1 Manipulation des clés

Créer une clé

```
SET nom "Alice"
```

Lire une clé

```
GET nom
```

Mettre à jour une clé

```
SET nom "Bob"
```

Supprimer une clé

```
DEL nom
```

4.2 Compteurs

```
SET visiteurs 0
INCR visiteurs
DECR visiteurs
```

5 Structures de Données Redis

5.1 Listes

```
RPUSH cours "Math" "Physique" "Chimie"
LRANGE cours 0 -1
LPOP cours
```

5.2 Ensembles

```
SADD noms "Alice" "Bob" "Charlie"
SMEMBERS noms
SREM noms "Charlie"
```

5.3 Hashmaps

```
HSET user:1 name "Alice" age 25 email "alice@example.com"
HGETALL user:1
HINCRBY user:1 age 1
```

5.4 Sorted Sets

```
ZADD scores 19 "Augustin"
ZADD scores 18 "Ines"
ZADD scores 20 "Philippe"

ZRANGE scores 0 -1
ZREVRANGE scores 0 -1
ZRANK scores "Augustin"
```

6 Comprendre les structures Redis et les commandes L/R

Redis fournit plusieurs structures de données, chacune optimisée pour un usage particulier. Les commandes utilisent souvent les préfixes **L** (Left) ou **R** (Right), ce qui indique l'extrémité de la structure sur laquelle l'opération agit. Cette logique est cruciale pour comprendre le comportement de Redis.

6.1 Listes : fonctionnement et logique L/R

Une liste Redis est une **liste chaînée** doublement liée. Elle permet des insertions et suppressions en temps constant à gauche ou à droite.

— **L** = Left = début de liste

— **R** = Right = fin de liste

Exemple : ajouter des éléments

```
RPUSH cours "Math" "Physique" "Chimie"
```

Le suffixe `R` signifie que les éléments sont ajoutés à **droite**.
État :

$$[Math,\ Physique,\ Chimie]$$

Extraire selon le côté

```
LPOP cours   # retire l'élément le plus à gauche
RPOP cours   # retire l'élément le plus à droite
```

Lire une partie de la liste

```
LRANGE cours 0 -1
```

`LRANGE` permet d'extraire une tranche. `0 -1` signifie « toute la liste ».

6.2 Ensembles : unicité et absence d'ordre

Les ensembles (*sets*) stockent des valeurs **uniques**, sans ordre.

Insertion

```
SADD noms "Alice" "Bob" "Charlie"
```

SADD ignore automatiquement les doublons.

Lire les éléments

```
SMEMBERS noms
```

Retourne les membres dans un ordre non garanti (structure hashée).

Suppression

```
SREM noms "Charlie"
```

6.3 Ensembles ordonnés (Sorted Sets)

Les *sorted sets* associent chaque membre à un **score** numérique. Les membres sont automatiquement triés selon ce score.

Insertion

```
ZADD scores 19 "Augustin"
ZADD scores 18 "Ines"
ZADD scores 20 "Philippe"
```

Lecture en ordre croissant

```
ZRANGE scores 0 -1
```

Lecture en ordre décroissant

```
ZREVRANGE scores 0 -1
```

Positionnement

```
ZRANK scores "Augustin"
```

Donne la position (index trié) de l'élément.

6.4 Hashmaps : regroupement d'attributs

Les hashmaps permettent d'associer plusieurs champs à une clé principale.

Insertion ou mise à jour

```
HSET user:1 name "Alice" age 25 email "alice@example.com"
```

Lecture complète

```
HGETALL user:1
```

Incrément numérique

```
HINCRBY user:1 age 1
```

6.5 Comparaison rapide

— **Listes** : ordonnées, manipuler en FIFO/LIFO avec LPOP/RPUSH.

— **Sets** : valeurs uniques, pas d'ordre.

— **Sorted Sets** : valeurs uniques + score + ordre trié.

— **Hashmaps** : structure clé–valeurs groupée au sein d'une même clé.

Ces structures permettent d'utiliser Redis pour du caching, de la file d'attente, des classements, des profils utilisateurs, etc.

7 Publication / Souscription (Pub/Sub)

7.1 S'abonner à un canal

```
SUBSCRIBE cours
```

7.2 Publier un message

```
PUBLISH cours "Un nouveau chapitre est disponible."
```

7.3 Abonnement par motif

```
PSUBSCRIBE cours_*
```

7.4 Publier

```
PUBLISH cours_info "TP Redis disponible."
```

8 Fonctionnalités Avancées de Redis

8.1 Transactions

```
MULTI
INCR x
INCR x
EXEC
```

8.2 Scripts Lua

```
EVAL "return redis.call('GET', KEYS[1])" 1 key1
```

8.3 HyperLogLog

```
PFADD visiteurs "User1" "User2" "User3"
PFCOUNT visiteurs
```

8.4 Clustering

Redis peut répartir ses données sur plusieurs nœuds pour la haute disponibilité. (Non manipulé dans ce TP.)