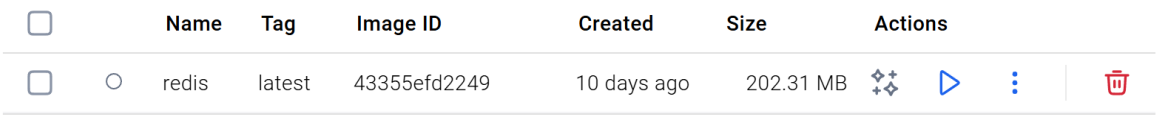
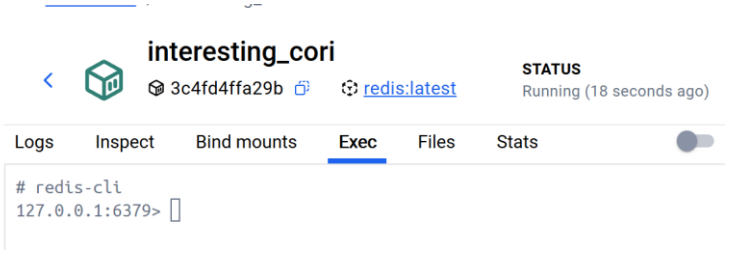
I. Redis

Redis est une base de données clé-valeur stockant les données en mémoire, ce qui la rend très rapide. Ce TP a pour objectif de découvrir son fonctionnement, d'apprendre à manipuler les différentes structures de données et à ajouter, modifier ou supprimer des clés. Il permet ainsi de comprendre les bases de Redis et son utilisation dans des applications nécessitant performance et simplicité.

J'utilise Docker Desktop pour ce TP



On utilise un client pour communiquer avec la BDD, on utilise l'utilitaire de ligne de commande CLI



On remarque notre serveur est en local, 127.0.0.1 et écoute sur le port 6379.

1. Manipulations de base

Créer une clé et lui attribuer une valeur : SET mykey myvalue

```
127.0.0.1:6379> SET prenom "Cath"
OK
```

Le serveur renvoie ok, la clé a été définie avec succès

Les différentes opérations que nous pouvons effectuer sont **CRUD** :

- Create, pour créer
- Read, pour lire
- Update, pour mettre à jour
- Delete, pour supprimer

On crée une deuxième clé :

```
127.0.0.1:6379> SET user:1234 "Arthur"
OK
```

Remarque : Redis est utilisé avec une autre BDD de type relationnel qui permet de sauvegarder les données sur un disque et pas uniquement sur la RAM. Il faut pour cela faire certaines configurations. Si une panne intervient, Redis ne pourra pas récupérer toutes les dernières valeurs si les configurations n'ont pas été faites.

Accéder à la **documentation Redis** : Docs

Pour récupérer la valeur : GET mykey

```
127.0.0.1:6379> GET user:1234
"Arthur"
127.0.0.1:6379>
```

Pour supprimer une clé : DEL mykey

```
127.0.0.1:6379> DEL user:1234
(integer) 1
```

Le serveur renvoie 1, pour confirmer que l'opération a été réalisé.

En cas de problème, le serveur renvoie 0 (exemple : essayer de supprimer une clé inexistante)

```
127.0.0.1:6379> DEL user:12345
(integer) 0
```

Exemple typique d'utilisation de Redis, compter le nombre de visiteurs d'un site web, qu'on sauvegarde dans une base de données de type clé-valeurs

Compteurs

Créer la clé-valeur : SET mykey initvalue

Incrémenter la valeur : INCR mykey

Décrémenter la valeur : DECR mykey

```
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
127.0.0.1:6379>
```

Remarque : Que se passe-t-il si plusieurs utilisateurs se connectent en même temps ?

Redis gère bien la concurrence car il est mono-thread :
il exécute une commande à la fois, dans l'ordre d'arrivée.

- ⇨ Donc chaque commande est atomique : elle ne peut pas être interrompue par une autre.
- ⇨ Même si deux utilisateurs se connectent exactement en même temps, Redis traite leurs requêtes l'une après l'autre, garantissant un résultat correct.

Pour des opérations comme INCR, le compteur sera toujours exact, sans risque de conflit.

Normalement Redis est utilisé pour les variables qui sont en RAM, pas sur le disque dur.
On peut définir une durée de vie pour une clé.

Pour avoir la durée de vie d'une clé : TTL mykey

- ⇨ Un nombre >= 0, le nombre de secondes restantes avant l'expiration
- ⇨ -1 si la durée de vie n'est pas définie
- ⇨ -2 si la clé n'existe pas

Redis stocke tout en RAM, comme énoncé précédemment, donc s'il n'y a plus d'espaces disponible, il y a 2 options :

- ⇨ Si une politique d'éviction est définie, il supprime automatiquement des clés

- Les moins utilisé (LRU)
- Les plus anciennes (TTL proche de l'expiration)
- Ou autres

Voici différentes politiques :

- volatile-lru : supprime les clés avec TTL, les moins utilisées
- allkeys-lru : supprime les clés les moins utilisées (même sans TTL)
- volatile-ttl : supprime d'abord celles qui expirent bientôt
- noeviction

⇨ Si la politique noeviction est définie, Redis va refuser les nouvelles écritures donc les commandes qui ajoutent ou modifient des données échouent, mais les lectures restent disponibles.

Pour définir le temps d'expiration d'une variable déjà existante : EXPIRE mykey nb_sec

```
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) 71
127.0.0.1:6379>
```

2. Structure de données

Les Listes

Définir une liste : RPUSH myList Element

```
127.0.0.1:6379> RPUSH mesCours "BDA"
(integer) 1
127.0.0.1:6379> RPUSH mesCours "Services Web"
(integer) 2
```

Récupérer les éléments d'une liste

Remarque : utiliser GET renvoie une erreur car on ne peut pas afficher une liste avec GET.

LRANGE myList 1stElement_indice Last_Element_indice

⇨ -1 pour Last_Element_indice si on veut afficher tout les éléments de la liste

⇨ 0 0 pour le premier élément (1 1 pr le second etc..)

```
127.0.0.1:6379> GET mesCours
(error) WRONGTYPE Operation against a key holding the wrong kind of value
127.0.0.1:6379> LRANGE mesCours 0 -1
1) "BDA"
2) "Services Web"
127.0.0.1:6379> LRANGE mesCours 0 0
1) "BDA"
127.0.0.1:6379> LRANGE mesCours 1 1
1) "Services Web"
127.0.0.1:6379>
```

Supprimer à droite/gauche d'une liste : LPOP (ou RPOP) myList [nb_element_a_supp]

⇨ [nb_element_a_supp] : optionnel, si on ne met rien ça ne supprimera qu'un seul élement

Remarque : Dans une liste on peut avoir plusieurs fois le même éléments, cela ne posera pas de problème.

Résumé :

• **LPUSH** ajoute un nouvel élément au début d'une liste.

• **RPUSH** ajoute un nouvel élément à la fin d'une liste.

• **LPOP** supprime et renvoie un élément au début d'une liste.

• **RPOP** supprime et renvoie un élément à la fin d'une liste.

• **LLEN** renvoie la longueur d'une liste.

• **LMOVE** déplace de manière atomique des éléments d'une liste vers une autre.

• **LRANGE** extrait une plage d'éléments d'une liste.

• **LTRIM** réduit une liste à la plage d'éléments spécifiée.

Les ensembles (set)

Caractéristiques :

⇨ Chaque élément est unique
⇨ Les ensembles ne sont pas ordonnées
⇨ Opérations très rapides
⇨ Opérations d'ensemble sur Redis (union, intersection)

Déclarer un ensemble : SADD mySet member

```
127.0.0.1:6379> SADD utilisateurs "Cath"
(integer) 1
127.0.0.1:6379> SADD utilisateurs "Arthur"
(integer) 1
127.0.0.1:6379> SADD utilisateurs "Ethan"
(integer) 1
127.0.0.1:6379> SADD utilisateurs "Fz"
(integer) 1
```

Si l'on essaye d'ajouter un élément déjà présent, l'opération ne s'effectue pas et nous recevons un zéro, à cause de l'unicité des éléments.

```
127.0.0.1:6379> SADD utilisateurs "Fz"
(integer) 0
```

Pour afficher les éléments d'un set : SMEMBERS mySet

```
127.0.0.1:6379> SMEMBERS utilisateurs
1) "Cath"
2) "Arthur"
3) "Ethan"
4) "Fz"
```

Supprimer un élément : SREM mySet member

```
127.0.0.1:6379> SREM utilisateurs "Fz"
(integer) 1
```

Opération Union : SUNION mySet1 mySet2

On crée un deuxième ensemble pour faire l'union

```
127.0.0.1:6379> SADD utilisateurs2 "Sarah"
(integer) 1
127.0.0.1:6379> SADD utilisateurs2 "Dima"
(integer) 1
```

```
127.0.0.1:6379> SUNION utilisateurs utilisateurs2
1) "Fz"
2) "Ethan"
3) "Arthur"
4) "Sarah"
5) "Cath"
6) "Dima"
```

Intersection : SINTER mySet1 mySet2

```
127.0.0.1:6379> SADD utilisateurs2 "Ethan"
(integer) 1
127.0.0.1:6379> SINTER utilisateurs utilisateurs2
1) "Ethan"
```

Résumé :

- SADD key value… → Ajoute un ou plusieurs éléments dans un ensemble
- SREM key value… → Supprime un ou plusieurs éléments
- SISMEMBER key value → Vérifie si un élément est présent
- SMEMBERS key → Retourne tous les éléments de l'ensemble
- SCARD key → Donne le nombre d'éléments dans l'ensemble
- SUNION key1 key2… → Union de plusieurs ensembles
- SINTER key1 key2… → Intersection de plusieurs ensembles
- SDIFF key1 key2… → Différence entre ensembles (éléments de key1 qui ne sont pas dans key2)

Sets ordonnés

Caractéristiques :

- ⇨ Chaque élément à un score (entier ou flottant)
- ⇨ Les éléments sont triés automatiquement par un score
- ⇨ Pas de doublons, élément unique
- ⇨ Accès rapide par rang

Créer un set ordonné : ZADD myOrdSet score member

```
127.0.0.1:6379> ZADD score4 19 "Arthur"
(integer) 1
127.0.0.1:6379> ZADD score4 18 "Fz"
(integer) 1
127.0.0.1:6379> ZADD score4 17 "Ethan"
(integer) 1
127.0.0.1:6379> ZADD score4 13 "Cath"
(integer) 1
```

Pour récupérer les éléments d'un set ordonné :

ZRANGE myOrdSet 1stElement_indice Last_Element_indice

- ⇨ -1 pour Last_Element_indice si on veut afficher tous les éléments de la liste
- ⇨ 0 1 pour les 2 premiers éléments etc…

```
127.0.0.1:6379> ZRANGE score4 0 -1
1) "Cath"
2) "Ethan"
3) "Fz"
4) "Arthur"
```

On remarque que les éléments ont été renvoyé par score croissant

Si on veut renvoyer par ordre décroissant :

ZREVRANGE myOrdSet 1stElement_indice Last_Element_indice

```
127.0.0.1:6379> ZREVRANGE score4 0 -1
1) "Arthur"
2) "Fz"
3) "Ethan"
4) "Cath"
```

Connaître la position d'un élément : ZRANK mySetOrd member

(La liste commence de 0, et c'est toujours dans l'ordre croissant)

```
127.0.0.1:6379> ZRANK   score4   "Fz"
(integer) 2
```

Dans les cas pratiques, si les informations doivent être récupéré dans le cadre de calcul assez fréquent, comme classer des utilisateurs par rapport à leur score. Les

informations seront déportées d'une base de données relationnelles vers Redis (qui joue donc le rôle du cache)

Pour utiliser une donnée (lecture, modification, calcul), elle doit d'abord être chargée en mémoire RAM. Le disque ne lit jamais uniquement la donnée demandée, mais un bloc complet.

• Taille d'un bloc : généralement 4096 octets (4 Ko), soit 512 octets × 8.
• Même pour lire 1 octet, le système charge 4 Ko en RAM.

Performances :
• Accès RAM : 10^{-8} à 10^{-7} secondes.
• Accès disque (HDD) : 10^{-2} secondes.

Conclusion : un accès disque est environ 1 000 000 fois plus lent qu'un accès RAM. Le débit de la RAM est 20 à 40 fois supérieur à celui d'un disque classique.

HASH

Structure qui permet de stocker plusieurs champs et valeurs sous une même clé.

Définir un hash : HSET key field value

```
127.0.0.1:6379> HSET user:11 username "ccath"
(integer) 1
127.0.0.1:6379> HSET user:11 age 23
(integer) 1
127.0.0.1:6379> HSET user:11 email ccali@gmail.com
(integer) 1
```

Récupérer tout les champs et valeurs : HGETALL key

```
127.0.0.1:6379> HGETALL user:11
1) "username"
2) "ccath"
3) "age"
4) "23"
5) "email"
6) "ccali@gmail.com"
```

Définir un hash avec tous les éléments : HMSET key field value ...

```
127.0.0.1:6379> HMSET user:4 username "Arthur" age 19 email arthurpokemon@gmail.com
OK
127.0.0.1:6379> HGETALL user:4
1) "username"
2) "Arthur"
3) "age"
4) "19"
5) "email"
6) "arthurpokemon@gmail.com"
```

Incrémenter un champ numérique : HINCRBY key field number

```
127.0.0.1:6379> HINCRBY user:4 age 4
(integer) 23
127.0.0.1:6379> HGETALL user:4
1) "username"
2) "Arthur"
3) "age"
4) "23"
5) "email"
6) "arthurpokemon@gmail.com"
```

Récupérer la valeur d'un champ : HGET key field

Récupérer toutes les valeurs : HVALS key

```
127.0.0.1:6379> HGET user:4 age
"23"
127.0.0.1:6379> HVALS user:4
1) "Arthur"
2) "23"
3) "arthurpokemon@gmail.com"
```

Résumé :

- HSET key field value : ajoute ou modifie un champ.
- HMSET key field value ... : ajoute plusieurs champs en une seule commande.
- HGET key field : récupère la valeur d'un champ.
- HGETALL key : récupère tous les champs et valeurs.
- HINCRBY key field number : incrémente un champ numérique.
- HVALS key : récupère toutes les valeurs.

Pub/Sub

Publish/Subscribe : système de messagerie en temps réel

Caractéristiques :

- ⇨ Communication temps réel
- ⇨ Basée sur des canaux (channels)
- ⇨ Pas de stockage des messages
- ⇨ Léger et rapide
- ⇨ Adapté pour notifications, chat, etc

On ouvre un deuxième terminal de commande (le client 1 est le terminal en mode sombre, client 2 terminal clair)

```
# redis-cli
127.0.0.1:6379> subscribe mescours user:1
1) "subscribe"
2) "mescours"
3) (integer) 1
1) "subscribe"
2) "user:1"
3) (integer) 2
Reading messages... (press Ctrl-C to quit or any key to type command)
```

On publie un nouveau message :

```
127.0.0.1:6379> PUBLISH mescours "un nouveau cours sur mongoDB"
(integer) 1
127.0.0.1:6379>
```

```
docker  exec -it 3c4fd4ffa29b25b850f724478497d28f4b30c5370d0227be406b69d6c23293cc /bin/sh
# redis-cli
127.0.0.1:6379> subscribe mescours user:1
1) "subscribe"
2) "mescours"
3) (integer) 1
1) "subscribe"
2) "user:1"
3) (integer) 2
1) "message"
2) "mescours"
3) "un nouveau cours sur mongoDB"
Reading messages... (press Ctrl-C to quit or any key to type command)
```

On peut voir que la personne abonnée a reçu le message en temps réel

Envoyer un message à un utilisateur : PUBLISH username message

```
127.0.0.1:6379> PUBLISH mescours "un nouveau cours sur mongoDB"
(integer) 1
127.0.0.1:6379> PUBLISH user:1 "Bonjour user 1"
(integer) 1
127.0.0.1:6379>
```

```
docker  exec -it 3c4fd4ffa29b25b850f724478497d28f4b30c5370d0227be406b69d6c23293cc /bin/sh
2) "mescours"
3) "un nouveau cours sur mongoDB"
1) "message"
2) "user:1"
3) "Bonjour user 1"
Reading messages... (press Ctrl-C to quit or any key to type command)
```

Remarque : On peut s'inscrire à plusieurs canaux, par exemple tous les canaux commençant par une chaîne de caractères.

PSUBSCRIBE pattern

```
127.0.0.1:6379> PSUBSCRIBE mes*
1) "psubscribe"
2) "mes*"
3) (integer) 1
```

```
127.0.0.1:6379> PUBLISH mesnotes "Une nouvelle note est arrivée"
(integer) 1
```

```
docker  exec -it 3c4fd4ffa29b25b850f724478497d28f4b30c5370d0227be406b69d6c23293cc /bin/sh
2) "mes*"
3) (integer) 1
1) "pmessage"
2) "mes*"
3) "mesnotes"
4) "Une nouvelle note est arriv\xc3\xa9e"
Reading messages... (press Ctrl-C to quit or any key to type command)
```

Résumé :

- SUBSCRIBE channel : s'abonne à un canal.
- PSUBSCRIBE pattern : s'abonne à un ensemble de canaux via un motif.
- UNSUBSCRIBE channel : se désabonne d'un canal.
- PUNSUBSCRIBE pattern : se désabonne d'un motif.
- PUBLISH channel message : publie un message dans un canal.
- PUBSUB CHANNELS : liste les canaux actifs.
- PUBSUB NUMSUB channel : affiche le nombre d'abonnés d'un canal.
- PUBSUB NUMPAT : affiche le nombre total de motifs actifs.

Pour avoir toutes les clés sauvegardées sur cette session : KEYS *

```
127.0.0.1:6379> KEYS *
1) "user:11"
2) "user11"
3) "score4"
4) "prenom"
5) "demo"
6) "utilisateurs2"
7) "utilisateurs"
8) "1mars"
9) "user:4"
```

Redis met a disposition des utilisateurs 16 base de données.

Par défaut on est connecté sur la base de donnée 0

Si on veut changer de base de données : SELECT nb_base

```
127.0.0.1:6379> SELECT 1
OK
127.0.0.1:6379[1]> KEYS *
(empty array)
127.0.0.1:6379[1]>
```

Reprise sur panne : En cas de panne, Redis perd toutes les données récentes qui n'ont pas encore été sauvegardées sur le disque. Par défaut, seule une partie des données est persistée, donc il est nécessaire de configurer correctement les mécanismes de sauvegarde pour éviter ou réduire la perte de données lors d'une interruption brutale du serveur.

Conclusion :

Cette première partie nous a permis de comprendre les principales structures de données de Redis (listes, ensembles, sets ordonnés, hash) et leurs commandes essentielles. Nous avons vu que Redis est extrêmement rapide grâce au stockage en mémoire, mais que cela implique de configurer la persistance pour éviter la perte de données en cas de panne. Nous avons également découvert le fonctionnement du Pub/Sub pour la communication en temps réel.