Prénom: Abdelillah
Nom: SELHI
N°étudiant: 12206456
Groupe: INFOA3

Introduction

Redis est un moteur de stockage en mémoire conçu pour offrir une latence extrêmement faible. Il fournit un ensemble riche de structures de données natives qui vont bien au-delà du simple couple clé-valeur, comme les listes et les ensembles En s'exécutant principalement en RAM et en traitant les commandes de manière atomique, il permet d'atteindre des performances difficiles à égaler pour la mise en cache, la gestion de sessions, la distribution d'événements.

Docker est une plateforme de conteneurisation qui encapsule des applications ainsi que leurs dépendances dans des environnements isolés, reproductibles et portables. Cette isolation garantit que l'exécution d'un service reste stable quel que soit l'hôte, tandis que la portabilité du format de conteneur facilite le déploiement continu, les tests, et la montée en charge. Utiliser Redis via Docker permet d'éviter toute installation locale complexe, de versionner facilement son environnement et de disposer d'un cycle de vie maîtrisé grâce aux commandes standards de Docker pour l'exécution, la persistance via volumes, la supervision ou le réseau.

1- INSTALLATION

A. INSTALLER DOCKER

sudo apt update
sudo apt install docker.io
sudo systemctl enable --now docker
docker --version //vérifier la version du docker

B. RÉCUPÉRER ET RUN L'IMAGE REDIS

docker run redis

C. Lancer redis-cli sur ce container

a. Récupérer l'ID du container avec **"docker ps"**
b. Lancer la commande:
docker exec -it c213d0461d64 redis-cli

D. Comment créer une clé valeur:

Lancer la commande **SET clé val**

```
127.0.0.1:6379> SET demo "Bonjour"
OK
```

```
127.0.0.1:6379> set user:1234 "Abdelillah"
OK
```

E. Comment récupérer la valeur d'une clé:

C'est avec la commande **GET clé**

```
127.0.0.1:6379> get user:1234
"Abdelillah"
```

F. Comment supprimer une clé:valeur:

Lancer la commande **del clé**
Réponse: 1 si la clé est supprimée, 0 sinon (celà peut être dû à une clé inexistante)

```
127.0.0.1:6379> del user:1234
(integer) 1
127.0.0.1:6379> del user:12345
(integer) 0
```

G. Comment incrémenter la valeur d'un clé entière:

Lancer la commande **incr clé**

```
127.0.0.1:6379> set nb 0
OK
127.0.0.1:6379> incr nb
(integer) 1
127.0.0.1:6379> incr nb
(integer) 2
127.0.0.1:6379> incr nb
(integer) 3
127.0.0.1:6379>
```

et si deux processus essaient d'incrémenter la valeur en même temps?
=> REDIS gère la concurrence.

Pour décrémenter la valeur de la clé, il suffit de lancer la commande "decr clé"

```
127.0.0.1:6379> set nb 0
OK
127.0.0.1:6379> incr nb
(integer) 1
127.0.0.1:6379> incr nb
(integer) 2
127.0.0.1:6379> incr nb
(integer) 3
127.0.0.1:6379> decr nb
(integer) 2
127.0.0.1:6379> decr nb
(integer) 1
```

H. Redis stocke principalement les données en mémoire vive pour offrir des temps d'accès extrêmement rapides. Selon la configuration, il peut également écrire sur disque pour assurer la persistance. Redis permet aussi d'attribuer une durée de vie à une clé grâce à un compteur de type TTL (Time To Live): une fois cette durée écoulée, la clé est automatiquement supprimée, ce qui facilite la gestion de données temporaires et optimise l'usage de la mémoire.

- **Pour avoir la durée de vie d'une clé**: lancer la commande: **TTL clé.**

```
127.0.0.1:6379> TTL nb
(integer) -1
```

(-1) veut dire que la durée de vie est illimitée et que la clé est gardée en RAM, jusqu'à ce que cette dernière soit pleine et que REDIS commence à supprimer certaines clés.

- **Pour attribuer une durée de vie à une clé**, lancer la commande: **expire clé nbsecondes**

```
127.0.0.1:6379> expire nb 500
(integer) 1
127.0.0.1:6379>
```

I. Peut-on avoir des collections de données plus complexes dans Redis?

Oui, Redis offre une multitude de types de collections telles que les listes et les ensembles.

Exemple: listes.

- **Pour créer une liste**: lancer la commande **RPUSH nom elt,** cette commande crée la liste et insère elt.

```
127.0.0.1:6379> RPUSH list 1
(integer) 1
127.0.0.1:6379> RPUSH list 2
(integer) 2
127.0.0.1:6379> RPUSH list 3
(integer) 3
```

- **Pour afficher la liste** on utilise **LRANGE i j,** où i est l'indice du début et j l'indice du dernier élément à afficher (pour afficher toute la liste: utiliser i=0 et j=-1)

```
127.0.0.1:6379> LRANGE list 0 -1
1) "1"
2) "2"
3) "3"
127.0.0.1:6379> LRANGE list 0 1
1) "1"
2) "2"
```

- **Pour supprimer un élément d'une liste**: utiliser **LPOP list** pour supprimer le premier élément à gauche et **RPOP list** pour supprimer le premier élément à droite.

```
127.0.0.1:6379> LPOP list
"1"
127.0.0.1:6379> RPOP list
"3"
```

Exemple: Set (ensemble) => pas de doublons.

- **Création et ajout d'éléments** (Ajouter un ou plusieurs éléments dans un set): **SADD myset value1 value2 value3**
- **Récupération des éléments: SMEMBERS myset**
- **Vérifier si une valeur est présente: SISMEMBER myset value1**
- **Supprimer un élément: SREM myset value1**
- **Supprimer un ou plusieurs sets: DEL myset**
- **Calculer l'intersection entre plusieurs sets: SINTER set1 set2.**
- **Obtenir la différence entre sets: DIFF set1 set2.**
- **Calculer l'union de deux ou plusieurs sets: SUNION set1 set2.**

EXEMPLE: Ordered Set

Ordered Set (ZSet) conserve les éléments **avec un score**, ce qui permet un **classement ordonné**. Les scores peuvent être entiers ou des flottants. Les valeurs restent uniques mais un même score peut être partagé.

- **Création et ajout d'éléments (ajouter un ou plusieurs éléments avec score): ZADD myzset 1 valeur1 2 valeur2 3 valeur3**
- **Récupération des éléments ordonnés (du plus petit score au plus grand): ZRANGE myzset 0 -1**
- **Récupération des éléments: ZRANGE myzset 0 -1**
- **Vérifier le score d'un élément: ZSCORE myzset valeur1**
- **Supprimer un élément spécifique: ZREM myzset valeur1**
- **Supprimer un ou plusieurs ordered sets: DEL myzset**
- **Obtenir le rang d'un élément (position dans l'ordre croissant)**
  ZRANK myzset valeur2
- **Obtenir le rang inverse (position en ordre décroissant)**
  ZREVRANK myzset valeur2
- **Obtenir une plage par score: ZRANGEBYSCORE myzset 0 10**
- **Obtenir une plage par score, en ordre inverse**
  ZREVRANGEBYSCORE myzset 10 0
- **Incrémenter le score d'un élément existant:**
  ZINCRBY myzset 5 valeur1
- **Taille totale du ordered set: ZCARD myzset**
- **Compter les éléments dans une plage de scores:**
  ZCOUNT myzset 0 10
- **Intersection entre plusieurs ordered sets (avec agrégation de scores): ZINTERSTORE result 2 zset1 zset2**
- **Union entre plusieurs ordered sets (somme ou autres méthodes d'agrégation): ZUNIONSTORE result 2 zset1 zset2**

EXEMPLE: HSET

HSET (Hash Set) : Permet de stocker des paires clé-valeur dans une structure de type hash (semblable à un dictionnaire). Chaque hash est identifié par une clé unique, et à l'intérieur, chaque champ possède une valeur associée.

- **Création et ajout de champs dans un hash :**
  HSET myhash field1 value1 field2 value2
- **Récupération de la valeur d'un champ : HGET myhash field1**
- **Récupération de tous les champs et valeurs : HGETALL myhash**
- **Incrémenter la valeur d'un champ numérique :**
  HINCRBY myhash field1 5
- **Récupération de toutes les valeurs : HVALS myhash**

J. PUB/SUB dans REDIS:

PUB/SUB est un mécanisme de messagerie appelé Publish/Subscribe par lequel les éditeurs envoient des messages sur des canaux et des consommateurs se subscribent à ces canaux**.**

- **S'abonner à un canal: SUBSCRIBE channel1 [channel2 ...]**
- **Publier un message: PUBLISH channel message**
- **Se désabonner: UNSUBSCRIBE [channel1 channel2 ...]**
- **Abonnement avec motif: PSUBSCRIBE pattern [pattern ...],** celà permet de s'abonner à tous les canaux dont le nom matche un motif.

K. Pour obtenir plus de commande, la documentation de REDIS est disponible: https://redis.io/docs/latest/commands/