Lab 3 Redis

redis

Professeur : BOUBCHIR, Larbi

Module : ADIF82

Table des matières

Introduction

Redis, acronyme de Remote Dictionary Server, est une base de données en mémoire réputée pour sa rapidité fulgurante. Créé en 2009 par Salvatore Sanfilippo, son développement a été motivé par le besoin d'une solution de stockage de données haute performance pour améliorer le système de scalabilité d'un site web italien. Depuis, elle est utilisée couramment comme système de gestion de base de données, cache et intermédiaire de messages, se distinguant par sa vitesse, sa simplicité et sa polyvalence.

L'un des atouts majeurs de Redis réside dans son stockage entièrement en mémoire, ce qui lui confère des vitesses de lecture et d'écriture exceptionnelles. Cette particularité rend Redis spécialement adapter pour les applications nécessitant des accès rapides, comme les systèmes de mise en cache, la gestion des sessions utilisateurs, ou encore les classements en temps réel. Par exemple, Twitter utilise Redis pour gérer les sessions utilisateur et les messages, exploitant sa rapidité pour des millions d'utilisateurs actifs.

En outre, cette technologie se distingue par la diversité des structures de données qu'elle supporte. Allant au-delà des simples chaînes, elle gère des listes, des ensembles, des ensembles triés, des hachages, des bitmaps, des hyperloglogs et des streams. Cette richesse structurelle offre une grande flexibilité aux développeurs pour la gestion et l'optimisation des données.

Sur le plan de la fiabilité et de la durabilité, Redis excelle grâce à des fonctionnalités telles que la réplication maître-esclave et des mécanismes de persistance des données (RDB et AOF). Ces outils garantissent la sécurité et la pérennité des informations stockées. De plus, le clustering Redis renforce sa capacité à maintenir un service continu, même en cas de défaillance d'un nœud.

Redis supporte également les transactions, permettant l'exécution atomique de séquences de commandes, une caractéristique essentielle pour maintenir l'intégrité des données dans des environnements complexes. Sa compatibilité avec de nombreux langages de programmation facilite son intégration dans divers projets et architectures logicielles.

Enfin, l'extensibilité est un autre atout significatif de Redis. Les modules Redis permettent aux utilisateurs d'ajouter des commandes et fonctionnalités personnalisées, adaptant la base de données à leurs besoins spécifiques. Cette flexibilité rend Redis extrêmement utile dans divers scénarios, tels que la mise en cache pour accélérer les applications, la gestion de sessions web, les systèmes de file d'attente de messages, l'analyse de données en temps réel, et les systèmes de recommandations personnalisées.

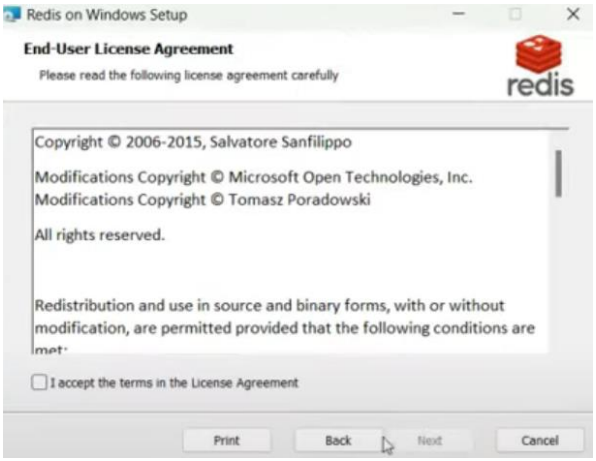
Ainsi, comparée à d'autres technologies comme Memcached, Redis offre une plus grande diversité de structures de données et de fonctionnalités, comme la persistance et le clustering, ce qui la rend adaptée à une plus large gamme d'applications, allant de la mise en cache simple à des solutions de stockage de données complexes et hautement disponibles.

Installation de Redis

1-Se rendre sur le site de Redis **Redis**

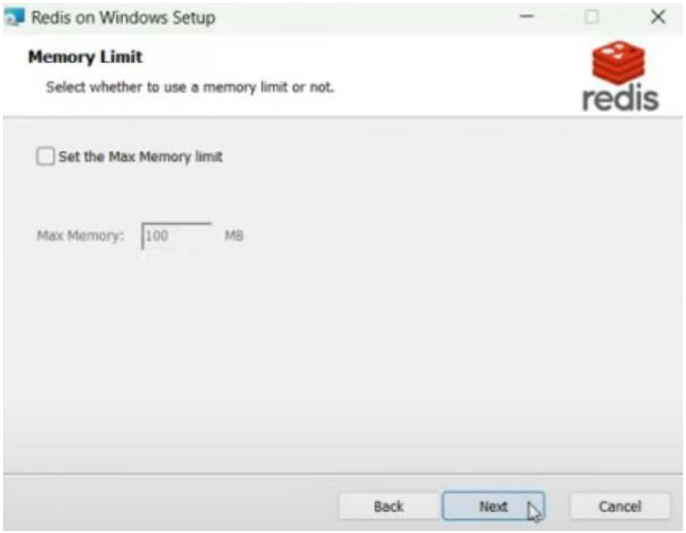
2-Download le fichier d'installation .msi de Redis.

3-Suivre le Setup d'installation



Nous pouvons choisir le path d'installation de Redis.

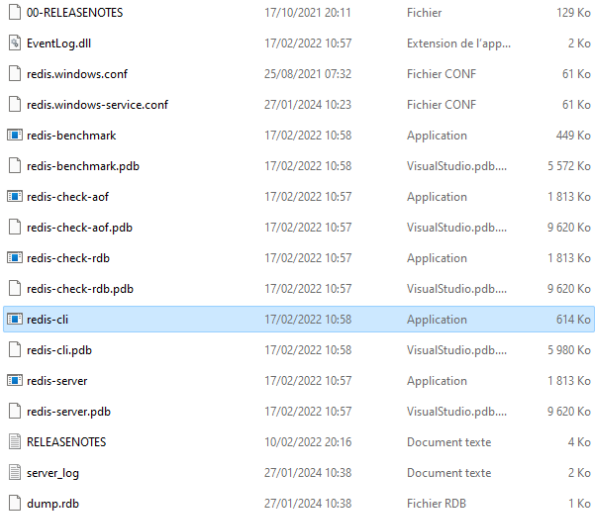
Il est possible de modifier le numéro de port utilisé par Redis.



Nous avons la possibilité de configurer la quantité maximale de RAM que nous souhaitons allouer à Redis.

L'installation de Redis est terminée.

4-Aller dans les fichiers ou se trouve Redis



00-RELEASENOTES	17/10/2021 20:11	Fichier	129 Ko
EventLog.dll	17/02/2022 10:57	Extension de l'app...	2 Ko
redis.windows.conf	25/08/2021 07:32	Fichier CONF	61 Ko
redis.windows-service.conf	27/01/2024 10:23	Fichier CONF	61 Ko
redis-benchmark	17/02/2022 10:58	Application	449 Ko
redis-benchmark.pdb	17/02/2022 10:58	VisualStudio.pdb....	5 572 Ko
redis-check-aof	17/02/2022 10:57	Application	1 813 Ko
redis-check-aof.pdb	17/02/2022 10:57	VisualStudio.pdb....	9 620 Ko
redis-check-rdb	17/02/2022 10:57	Application	1 813 Ko
redis-check-rdb.pdb	17/02/2022 10:57	VisualStudio.pdb....	9 620 Ko
redis-cli	17/02/2022 10:58	Application	614 Ko
redis-cli.pdb	17/02/2022 10:58	VisualStudio.pdb....	5 980 Ko
redis-server	17/02/2022 10:57	Application	1 813 Ko
redis-server.pdb	17/02/2022 10:57	VisualStudio.pdb....	9 620 Ko
RELEASENOTES	10/02/2022 20:16	Document texte	4 Ko
server_log	27/01/2024 10:38	Document texte	2 Ko
dump.rdb	27/01/2024 10:38	Fichier RDB	1 Ko

5-Ouvrire le redis-cli pour voir si l'installation à bien fonctionné

```
127.0.0.1:6379> set name sumit
OK
127.0.0.1:6379> get name
"sumit"
```

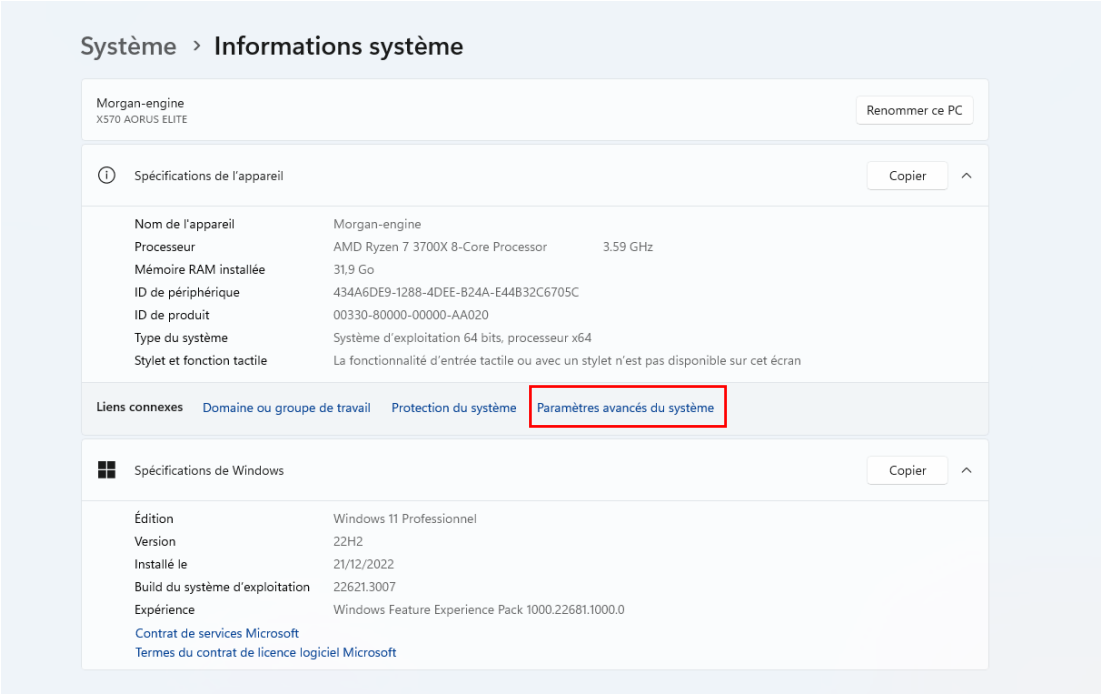
L'utilisation de Redis fonctionne.

6-Ouvrire le CMD (Command Prompt) en tant qu'administrateur

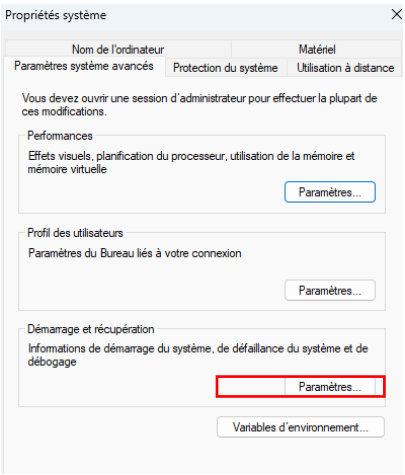
```
C:\Windows\System32>redis-server
'redis-server' n'est pas reconnu en tant que commande interne
ou externe, un programme exécutable ou un fichier de commandes.

C:\Windows\System32>
```

L'invite de commande (cmd) ne reconnaît pas Redis, car Redis n'a pas été défini dans les variables d'environnement.

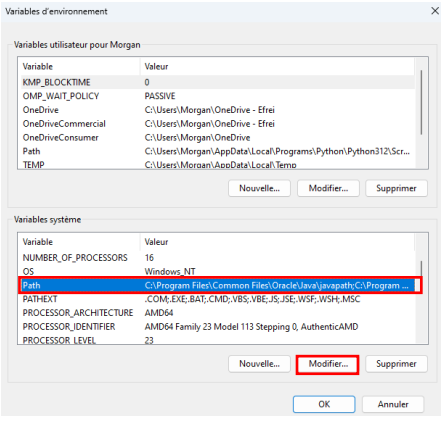
7-Définir Redis dans les variables d'environnement



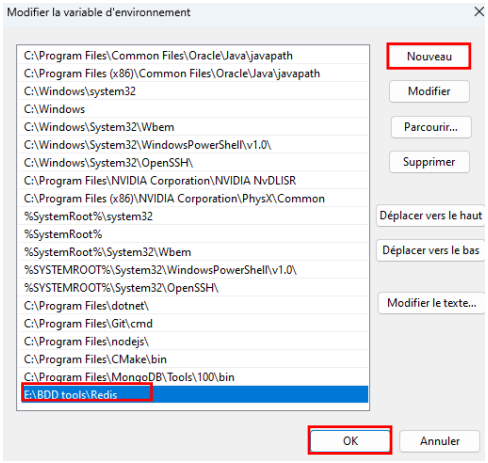
Allez dans les informations système et cliquez sur 'Paramètres avancés du système'.



Cliquez sur 'Variables d'environnement'.



Accédez aux 'Variables système', puis cliquez sur 'Path' et sélectionnez 'Modifier'.



Cliquez sur 'Nouveau', puis copiez le chemin où Redis est installé. Ensuite, cliquez sur 'OK', fermez les paramètres système et redémarrez l'invite de commande en tant qu'administrateur.

8-Test de Redis

```
C:\Windows\System32>redis-server
[17416] 27 Jan 11:16:37.219 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
[17416] 27 Jan 11:16:37.219 # Redis version=5.0.14.1, bits=64, commit=ec77f72d, modified=0, pid=17416, just started
[17416] 27 Jan 11:16:37.219 # Warning: no config file specified, using the default config. In order to specify a config
file use redis-server /path/to/redis.conf
[17416] 27 Jan 11:16:37.221 # Could not create server TCP listening socket *:6379: bind: Une opÛration a ÚtÚ tentÛe sur
autre chose quÆun socket.

C:\Windows\System32>
```

Dans le nouveau terminal de commande (cmd), saisissez la commande 'redis-server'. Le serveur sera lancé

```
C:\Windows\System32>redis-cli
127.0.0.1:6379> ping
PONG
127.0.0.1:6379>
```

Maintenant, ouvrez un autre terminal de commande en tant qu'administrateur, puis saisissez la commande 'redis-cli' pour vous connecter au client Redis. Exécutez le test 'ping' pour vérifier si tout fonctionne correctement.

```
127.0.0.1:6379> set key value [expiration EX seconds|PX milliseconds] [NX|XX]
```

Nous pouvons constater que lorsque nous tapons la commande 'set' dans Redis, il nous propose immédiatement d'entrer une paire : Clé / Valeur. Testons si cela fonctionne :

```
127.0.0.1:6379> set name sumit
OK
127.0.0.1:6379> set channel playjava
OK
127.0.0.1:6379> get name
"sumit"
127.0.0.1:6379> get channel
"playjava"
127.0.0.1:6379>
```

Nous pouvons voir que les paires Clé/Valeur : 'name/sumit' et 'channel/playjava' ont bien été enregistrées. Si nous voulons rechercher nos valeurs en utilisant les clés, il nous suffit de taper la commande 'get' suivie de la clé. Nous pouvons constater que 'get name' nous renvoie 'sumit' et 'get channel' nous renvoie 'playjava' (c'est-à-dire les valeurs associées à chacune des clés enregistrées précédemment). L'installation est donc terminée, Redis est fonctionnel et prêt à être utilisé.

Création d'une base de données avec Redis

Le fonctionnement des BDD sur Redis

Par défaut, lorsque l'on lance Redis, nous disposons de 16 bases de données disponibles. Lorsque nous nous connectons au CLI de Redis en utilisant la commande 'redis-cli', nous nous connectons à la base de données 0.

```
C:\Windows\System32>redis-cli
127.0.0.1:6379> KEYS *
 1) "users:456"
 2) "score"
 3) "user:1"
 4) "name"
 5) "user:123"
 6) "days:123"
 7) "sidebar"
 8) "user:456"
 9) "users"
10) "users2"
11) "example"
12) "name:"
13) "channel"
127.0.0.1:6379>
```

Ici, nous pouvons voir que toutes ces clés ont été sauvegardées dans la base de données 0.

```
127.0.0.1:6379> SELECT 1
OK
127.0.0.1:6379[1]> KEYS *
(empty list or set)
127.0.0.1:6379[1]>
```

Si nous souhaitons changer de base de données, il nous suffit d'utiliser l'opérateur SELECT suivi de l'index de la base de données que nous souhaitons utiliser. Ici, nous nous sommes connectés à la base de données 1. Nous pouvons constater que aucune clé n'a été créée sur cette base de données.

```
127.0.0.1:6379[1]> SELECT 45
(error) ERR DB index is out of range
127.0.0.1:6379[1]>
```

Étant donné que le nombre de bases de données implémentées par défaut par Redis est de 16, si nous voulons nous connecter à une base de données supérieure à 16.

Création d'une BDD cas concret

```
127.0.0.1:6379[1]>
```

Pour cet exemple, nous allons travailler sur la base de données numéro 1, étant donné qu'elle est vide.

1. Caching

Objectif : Stocker des pages ou des données fréquemment demandées pour accélérer l'accès.

Structure de Données : Strings

```
127.0.0.1:6379[1]> SET page:home "<html>Contenu de la page d'accueil</html>"
OK
127.0.0.1:6379[1]> GET page:home
"<html>Contenu de la page d'accueil</html>"
127.0.0.1:6379[1]> EXPIRE page:home 3600
(integer) 1
127.0.0.1:6379[1]>
```

Stocker une page : SET page:home "<html>Contenu de la page d'accueil</html>"

Récupérer une page : GET page:home

Définir une expiration : EXPIRE page:home 3600 (expire après 1 heure)

2. Session Store

Objectif : Gérer les sessions utilisateur pour une application web.

Structure de Données : Hashes

```
127.0.0.1:6379[1]> HSET session:1234 userId 001 token abc123
(integer) 2
127.0.0.1:6379[1]> HGETALL session:1234
1) "userId"
2) "001"
3) "token"
4) "abc123"
127.0.0.1:6379[1]> DEL session:1234
(integer) 1
127.0.0.1:6379[1]>
```

Créer une session : HSET session:1234 userId 001 token abc123

Récupérer des informations de session : HGETALL session:1234

Supprimer une session : DEL session:1234

3. Message Broker

Objectif : Gérer les files d'attente de messages pour le traitement asynchrone.

Structure de Données : Lists pour une file d'attente simple, Streams pour des cas plus complexes.

```
127.0.0.1:6379[1]> LPUSH queue:message "Message 1"
(integer) 1
127.0.0.1:6379[1]> RPOP queue:message
"Message 1"
127.0.0.1:6379[1]> XADD stream:message * body "Hello World"
"1706361590832-0"
127.0.0.1:6379[1]> XREAD COUNT 1 STREAMS stream:message 0
1) 1) "stream:message"
   2) 1) 1) "1706361590832-0"
         2) 1) "body"
            2) "Hello World"
127.0.0.1:6379[1]>
```

Ajouter un message à la file : LPUSH queue:message "Message 1"

Consommer un message : RPOP queue:message

Pour les Streams, ajouter un message : XADD stream:message * body "Hello World"

Lire le message : XREAD COUNT 1 STREAMS stream:message 0

4. Stockage de Données en Temps Réel

Objectif : Mettre à jour et récupérer des données en temps réel, comme les scores en direct.

Structure de Données : Sorted Sets

```
127.0.0.1:6379[1]> ZADD scoreboard 100 "Joueur1"
(integer) 1
127.0.0.1:6379[1]> ZRANGE scoreboard 0 -1 WITHSCORES
1) "Joueur1"
2) "100"
127.0.0.1:6379[1]>
```

Ajouter/mettre à jour un score : ZADD scoreboard 100 "Joueur1"

Récupérer le classement : ZRANGE scoreboard 0 -1 WITHSCORES

5. Gestion des Sets

Objectif : Stocker des collections uniques d'éléments, comme des tags ou des membres d'un groupe.

```
127.0.0.1:6379[1]> SADD tags "redis" "database" "cache"
(integer) 3
127.0.0.1:6379[1]> SISMEMBER tags "redis"
(integer) 1
127.0.0.1:6379[1]> SMEMBERS tags
1) "redis"
2) "cache"
3) "database"
127.0.0.1:6379[1]>
```

Ajouter des éléments à un set : SADD tags "redis" "database" "cache"

Vérifier si un élément est dans le set : SISMEMBER tags "redis"

Récupérer tous les éléments d'un set : SMEMBERS tags

6.Listes avec Durée de Vie Limitée

Objectif : Créer des listes de tâches ou de messages qui expirent après un certain temps.

```
127.0.0.1:6379[1]> LPUSH tempQueue "Task1"
(integer) 1
127.0.0.1:6379[1]> EXPIRE tempQueue 600
(integer) 1
127.0.0.1:6379[1]> TTL tempQueue
(integer) 566
127.0.0.1:6379[1]>
```

Ajouter à une liste avec expiration : LPUSH tempQueue "Task1"

Définir une expiration sur la liste : EXPIRE tempQueue 600 (expire après 10 minutes)

Voir le temps avant expiration de la liste : TTL tempQueue

7.Hashes avec Plusieurs Champs

Objectif : Stocker des objets avec plusieurs champs, comme des profils d'utilisateurs ou des configurations.

```
127.0.0.1:6379[1]> HMSET user:1001 name "John Doe" age "30" email "john@example.com"
OK
127.0.0.1:6379[1]> HMGET user:1001 name age
1) "John Doe"
2) "30"
127.0.0.1:6379[1]> HSET user:1001 age "31"
(integer) 0
127.0.0.1:6379[1]> HMGET user:1001
(error) ERR wrong number of arguments for 'hmget' command
127.0.0.1:6379[1]> HMGET user:1001 age
1) "31"
127.0.0.1:6379[1]>
```

Créer un profil utilisateur : HMSET user:1001 name "John Doe" age "30" email "john@example.com"

Récupérer des champs spécifiques : HMGET user:1001 name age

Mettre à jour un champ : HSET user:1001 age "31"

8.Incrémentation et Décrémentation de Compteurs

Objectif : Gérer des compteurs, comme le nombre de visites sur une page.

```
127.0.0.1:6379[1]> SET pageViews:home 0
OK
127.0.0.1:6379[1]> INCR pageViews:home
(integer) 1
127.0.0.1:6379[1]> GET pageViews:home
"1"
127.0.0.1:6379[1]>
```

Initialiser un compteur : SET pageViews:home 0

Incrémenter le compteur : INCR pageViews:home

Récupérer la valeur du compteur : GET pageViews:home

9.Pub/Sub pour la Communication en Temps Réel

Objectif : Mettre en place un système de publication et d'abonnement pour les notifications ou les mises à jour en temps réel.

```
127.0.0.1:6379[1]> PUBLISH updates "New update available"
(integer) 1
127.0.0.1:6379[1]>
```

```
C:\Windows\System32>redis-cli
127.0.0.1:6379> SELECT 1
OK
127.0.0.1:6379[1]> SUBSCRIBE updates
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "updates"
3) (integer) 1
```

Publier un message : PUBLISH updates "New update available"

S'abonner à un canal : SUBSCRIBE updates (à exécuter dans un autre client Redis)

10.Géolocalisation

Objectif : Stocker et interroger des données basées sur la localisation.

```
127.0.0.1:6379[1]> GEOADD locations 2.294481 48.858370 "Eiffel Tower"
```

```
127.0.0.1:6379[1]> GEOADD locations 2.336443 48.860611 "Louvre"
```

```
127.0.0.1:6379[1]> GEODIST locations "Eiffel Tower" "Louvre" m
"3080.6480"
```

Ajouter des coordonnées géographiques : GEOADD locations 2.294481 48.858370 "Eiffel Tower"

Trouver la distance entre deux lieux : GEODIST locations "Eiffel Tower" "Louvre" m

11.Utilisation Avancée des Streams

Objectif : Gérer un flux de données pour des applications telles que les journaux d'événements ou les fils d'actualité.

```
127.0.0.1:6379[1]> XADD stream:message * body "Second message"
"1706362676796-0"
```

```
127.0.0.1:6379[1]> XADD stream:message * body "Third message"
"1706362687156-0"
```

```
127.0.0.1:6379[1]> XREAD COUNT 2 STREAMS stream:message 0
1) 1) "stream:message"
   2) 1) 1) "1706361590832-0"
         2) 1) "body"
            2) "Hello World"
      2) 1) "1706362676796-0"
         2) 1) "body"
            2) "Second message"
127.0.0.1:6379[1]> XREAD COUNT 3 STREAMS stream:message 0
1) 1) "stream:message"
   2) 1) 1) "1706361590832-0"
         2) 1) "body"
            2) "Hello World"
      2) 1) "1706362676796-0"
         2) 1) "body"
            2) "Second message"
      3) 1) "1706362687156-0"
         2) 1) "body"
            2) "Third message"
```

Ajouter plusieurs messages au stream :

- XADD stream:message * body "Second message"
- XADD stream:message * body "Third message"

Lire les 2 derniers messages : XREAD COUNT 2 STREAMS stream:message 0

Lire les 3 derniers messages : XREAD COUNT 3 STREAMS stream:message 0

12.Transactions

Objectif : Exécuter un ensemble de commandes atomiquement.

```
127.0.0.1:6379[1]> MULTI
OK
127.0.0.1:6379[1]> INCR pageViews:home
QUEUED
127.0.0.1:6379[1]> INCR pageViews:home
QUEUED
127.0.0.1:6379[1]> EXEC
1) (integer) 2
2) (integer) 3
127.0.0.1:6379[1]>
```

Démarrer une transaction : MULTI

Exécuter des commandes dans la transaction :

- INCR pageViews:home
- INCR pageViews:home

Appliquer la transaction : EXEC

(integer) 2 et (integer) 3 indiquent les nouvelles valeurs du compteur **pageViews:home** après chaque incrément.

13.Sets et Opérations sur les Sets

Objectif : Utiliser des sets pour gérer des collections uniques et effectuer des opérations sur ces ensembles.

```
127.0.0.1:6379[1]> SADD users "Alice" "Bob" "Charlie"
(integer) 3
```

```
127.0.0.1:6379[1]> SINTER tags users
(empty list or set)
```

```
127.0.0.1:6379[1]> SUNION tags users
1) "cache"
2) "database"
3) "Charlie"
4) "redis"
5) "Alice"
6) "Bob"
```

```
127.0.0.1:6379[1]> SDIFF tags users
1) "redis"
2) "cache"
3) "database"
127.0.0.1:6379[1]>
```

Ajouter des éléments à un autre set : SADD users "Alice" "Bob" "Charlie"

Intersection de deux sets : SINTER tags users

Union de deux sets : SUNION tags users

Différence entre deux sets : SDIFF tags users

14.Listes avec Opérations de Blocage

Objectif : Utiliser des listes pour implémenter des files d'attente de travail avec attente bloquante.

```
127.0.0.1:6379[1]> LPUSH workQueue "Task2" "Task3"
(integer) 2
127.0.0.1:6379[1]> BRPOP workQueue 30
1) "workQueue"
2) "Task2"
127.0.0.1:6379[1]>
```

Ajouter des tâches à une autre liste : LPUSH workQueue "Task2" "Task3"

Récupérer une tâche avec blocage : BRPOP workQueue 30

15.Gestion de Clés et de Bases de Données

Objectif : Gérer efficacement les clés et les différentes bases de données Redis.

```
127.0.0.1:6379[1]> KEYS *
1) "tags"
2) "workQueue"
3) "user:1001"
4) "users"
5) "page:home"
6) "pageViews:home"
7) "scoreboard"
8) "stream:message"
9) "locations"
127.0.0.1:6379[1]> RENAME tempKey newTempKey
(error) ERR no such key
127.0.0.1:6379[1]> RENAME user:1001 user:101
OK
127.0.0.1:6379[1]> MOVE user:101 0
(integer) 1
127.0.0.1:6379[1]> SELECT 0
OK
127.0.0.1:6379> KEYS *
 1) "users:456"
 2) "score"
 3) "user:1"
 4) "name"
 5) "user:123"
 6) "user:101"
 7) "days:123"
 8) "sidebar"
 9) "user:456"
10) "users"
11) "users2"
12) "example"
13) "name:"
14) "channel"
127.0.0.1:6379>
```

Lister toutes les clés : KEYS *

Renommer une clé : RENAME user :1001 user :101

Déplacer une clé vers une autre base de données : MOVE user101 0

Basculer vers une autre base de données : SELECT 0

16.Agrégation avec Sorted Sets

Objectif : Utiliser les sorted sets pour faire des classements et des agrégations.

```
127.0.0.1:6379> ZADD playerScores 50 "Joueur2"
(integer) 1
127.0.0.1:6379> ZADD playerScores 70 "Joueur3"
(integer) 1
127.0.0.1:6379> ZRANK playerScores "Joueur1"
(nil)
127.0.0.1:6379> ZREVRANGE playerScores 0 2 WITHSCORES
1) "Joueur3"
2) "70"
3) "Joueur2"
4) "50"
127.0.0.1:6379>
```

Ajouter des scores pour différents joueurs :

- ZADD playerScores 50 "Joueur2"
- ZADD playerScores 70 "Joueur3"

Obtenir le classement des joueurs : ZRANK playerScores "Joueur1"

Obtenir les meilleurs scores : ZREVRANGE playerScores 0 2 WITHSCORES

17.Bitmaps pour Comptage Unique

Objectif : Utiliser les bitmaps pour le suivi et le comptage uniques, comme les utilisateurs actifs.

```
127.0.0.1:6379> SETBIT activeUsers 1001 1
(integer) 1
127.0.0.1:6379> BITCOUNT activeUsers
(integer) 1
127.0.0.1:6379>
```

Marquer un utilisateur comme actif : SETBIT activeUsers 1001 1

Compter le nombre d'utilisateurs actifs : BITCOUNT activeUsers

18.HyperLogLogs pour l'Estimation de Cardinalité

Objectif : Utiliser HyperLogLogs pour estimer le nombre d'éléments uniques dans un ensemble de données volumineux.

```
127.0.0.1:6379> PFADD hllUsers "user1"
(integer) 1
127.0.0.1:6379> PFADD hllUsers "user2"
(integer) 1
127.0.0.1:6379> PFCOUNT hllUsers
(integer) 2
127.0.0.1:6379>
```

Ajouter des éléments à un HyperLogLog :

- PFADD hllUsers "user1"
- PFADD hllUsers "user2"

Estimer le nombre d'éléments uniques : PFCOUNT hllUsers

19.Scripting avec Lua

Objectif : Écrire des scripts Lua pour des opérations personnalisées.

```
127.0.0.1:6379> EVAL "return redis.call('SET', 'key', 'value')" 0
OK
127.0.0.1:6379>
```

Utiliser EVAL pour exécuter un script Lua simple : EVAL "return redis.call('SET', 'key', 'value')" 0

Exploration des fonctionnalités de Redis

Library Redis: Commands | Redis

Fonctionnalité Principal :

Opération SET / GET :

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

Cette commande permet de créer une paire Clé/Valeur : 'demo' / 'Salut'. Le message 'OK' nous indique que la paire Clé/Valeur a bien été enregistrée.

```
127.0.0.1:6379> GET demo
"Salut"
127.0.0.1:6379>
```

Cette commande permet de récupérer la valeur associée à une clé. Dans ce cas, la valeur est 'Salut' et la clé est 'demo'

OPERATION INCR / DECR:

```
127.0.0.1:6379> SET user:1 "Morgan"
OK
127.0.0.1:6379> GET user:1
"Morgan"
127.0.0.1:6379>
```

Clé : 'user:1' / Valeur : 'Morgan

```
127.0.0.1:6379> SET days:123 0
OK
127.0.0.1:6379> INCR days:123
(integer) 1
127.0.0.1:6379> INCR days:123
(integer) 2
127.0.0.1:6379> GET days:123
"2"
127.0.0.1:6379>
```

La commande SET 'days:123' initialise la clé 'days:123' avec la valeur 0. Ensuite, la commande INCR 'days:123' est utilisée deux fois, ce qui incrémente la valeur de la clé 'days:123' de 1 à chaque fois, aboutissant finalement à une valeur de 2. Enfin, la commande GET 'days:123' récupère la valeur actuelle de la clé 'days:123', qui est 2.

L'avantage de ce système est que s'il y a beaucoup de concurrence, Redis met en place un mécanisme qui permet de ne pas bloquer les opérations et de gérer les incréments de manière séquentielle.

```
127.0.0.1:6379> DECR days:123
(integer) 1
127.0.0.1:6379>
```

De la même manière, si nous utilisons la commande DECR, nous allons alors décrémenter la valeur associée à une clé.

```
127.0.0.1:6379> INCR days:124
(integer) 1
127.0.0.1:6379>
```

Ici, nous choisissons d'incrémenter une valeur qui n'existe pas. Redis part donc du principe que nous commençons l'incrémentation à partir de 0. Redis est très efficace dans la gestion du cache et sa capacité à gérer automatiquement l'expiration des informations. Les données SET ne sont jamais supprimées par Redis, sauf si la limite de la RAM est atteinte.

Opération sur le temps de vie d'une variable :

```
127.0.0.1:6379> TTL days:124
(integer) -1
127.0.0.1:6379>
```

Cette commande nous permet de vérifier le temps de vie d'une clé à l'aide de l'opérateur TTL. Dans ce cas, la commande retourne -1, ce qui signifie que c'est une variable qui n'expire pas.

```
127.0.0.1:6379> EXPIRE days:124 120
(integer) 1
127.0.0.1:6379>
```

Cette commande permet de définir le temps de vie d'une variable. Dans ce cas, l'opérateur EXPIRE permet d'implémenter un temps de vie pour une clé en secondes, qui est ici de 120 secondes.

```
127.0.0.1:6379> TTL days:124
(integer) 56
127.0.0.1:6379>
```

Si nous revenons sur le temps de vie de notre clé, nous pouvons constater que le compte à rebours pour la durée de vie de la clé est en train de diminuer et est actuellement à 56 secondes.

```
127.0.0.1:6379> GET days:124
(nil)
127.0.0.1:6379>
```

Si nous essayons d'obtenir cette clé, nous pouvons constater qu'elle n'existe plus car son temps de vie s'est entièrement écoulé.

```
127.0.0.1:6379> SET name: "Carl"
OK
127.0.0.1:6379> EXPIRE name: 124
(integer) 1
127.0.0.1:6379> TTL name:
(integer) 108
127.0.0.1:6379> SET name: "Laurent"
OK
127.0.0.1:6379> TTL name:
(integer) -1
127.0.0.1:6379>
```

Ici, nous pouvons observer que lorsque nous appliquons une durée de vie à une clé, mais que par la suite nous modifions la valeur associée à cette clé, la durée de vie de la clé est automatiquement réinitialisée.

Opération DEL :

```
127.0.0.1:6379> DEL demo
(integer) 1
127.0.0.1:6379> GET demo
(nil)
127.0.0.1:6379>
```

Cette commande nous permet de supprimer une paire Clé/Valeur. Dans ce cas, nous utilisons DEL pour supprimer la clé 'demo' associée à la valeur 'Salut'. L'opération nous retourne 1 pour indiquer que la paire Clé/Valeur a bien été supprimée. Si nous essayons de récupérer la clé qui vient d'être supprimée, alors nous obtenons 'nil', ce qui signifie que la clé n'existe pas.

Opération sur les LIST :

```
127.0.0.1:6379> RPUSH example "Jungle"
(integer) 1
127.0.0.1:6379> LPUSH example "Montagne"
(integer) 2
127.0.0.1:6379>
```

La commande RPUSH 'example' 'Jungle' ajoute la valeur 'Jungle' à la fin d'une liste nommée 'example', ce qui est utile pour construire des collections de données ordonnées. De même, la commande LPUSH 'example' 'Montagne' ajoute la valeur 'Montagne' au début de la même liste.

```
127.0.0.1:6379> LRANGE example 0 -1
1) "Montagne"
2) "Jungle"
127.0.0.1:6379>
```

Cette commande nous permet de récupérer les éléments de notre liste 'example'. Dans ce cas, l'opérateur LRANGE est utilisé pour récupérer les éléments de la liste. Les indices 0 et -1 sont utilisés pour récupérer tous les éléments de la liste, c'est-à-dire du premier au dernier élément.

```
127.0.0.1:6379> LRANGE example 0 0
1) "Montagne"
127.0.0.1:6379>
```

Ici, nous récupérons uniquement l'élément d'indice 0 de la liste.

```
127.0.0.1:6379> LLEN example
(integer) 2
127.0.0.1:6379>
```

Cette commande nous permet de récupérer la taille d'une liste grâce à l'opérateur LLEN. Dans ce cas, notre liste 'example' contient 2 éléments.

```
127.0.0.1:6379> LPOP example
"Montagne"
127.0.0.1:6379> LRANGE example 0 -1
1) "Jungle"
127.0.0.1:6379>
```

Cette commande nous permet de supprimer le premier élément de la liste grâce à l'opérateur LPOP. Dans ce cas, nous pouvons constater que l'élément 'Montagne' a bien été supprimé de notre liste 'example'. Si nous vérifions l'ensemble de notre liste, il ne reste que 'Jungle'.

```
127.0.0.1:6379> RPOP example
"Jungle"
127.0.0.1:6379> LRANGE example 0 -1
(empty list or set)
127.0.0.1:6379>
```

```
127.0.0.1:6379> LLEN example
(integer) 0
127.0.0.1:6379>
```

Cette commande nous permet de supprimer le dernier élément d'une liste grâce à l'opérateur RPOP. Dans ce cas, nous pouvons constater que le dernier élément de notre liste 'Jungle' a bien été supprimé. Si nous vérifions tous les éléments de notre liste 'example', nous pouvons constater que celle-ci est maintenant vide.

```
127.0.0.1:6379> LPUSH example "Mer"
(integer) 1
127.0.0.1:6379> LPUSH example "Mer"
(integer) 2
127.0.0.1:6379> LPUSH example "Mer"
(integer) 3
127.0.0.1:6379> LPUSH example "Mer"
(integer) 4
127.0.0.1:6379> LPUSH example "Mer"
(integer) 5
127.0.0.1:6379> LRANGE example 0 -1
1) "Mer"
2) "Mer"
3) "Mer"
4) "Mer"
5) "Mer"
127.0.0.1:6379>
```

Dans une liste, il est possible d'ajouter plusieurs fois la même valeur.

Opération sur les SADD :

```
127.0.0.1:6379> SADD users "Ryan"
(integer) 1
127.0.0.1:6379> SADD users "Jules"
(integer) 1
127.0.0.1:6379> SADD users "Ryan"
(integer) 0
127.0.0.1:6379>
```

Pour éviter la redondance des valeurs, nous pouvons créer des ensembles de valeurs avec l'opérateur SADD. Dans ce cas, nous avons créé 2 utilisateurs : Ryan et Jules. Lorsque nous essayons d'ajouter un

troisième utilisateur nommé Ryan, qui porte le même nom que le premier, cela nous renvoie 0 pour indiquer qu'il est impossible d'ajouter l'utilisateur Ryan car il existe déjà.

```
127.0.0.1:6379> SMEMBERS users
1) "Ryan"
2) "Jules"
127.0.0.1:6379>
```

Cette commande nous permet de voir la liste des éléments dans un ensemble (SET) grâce à l'opérateur SMEMBERS. Dans ce cas, nous pouvons constater que la commande nous retourne tous les éléments de l'ensemble 'users'.

```
127.0.0.1:6379> SREM users "Jules"
(integer) 1
127.0.0.1:6379> SMEMBERS users
1) "Ryan"
127.0.0.1:6379>
```

Cette commande nous permet de supprimer un élément dans un SET grâce à l'opérateur SREM. Ici, nous pouvons voir que Jules a été supprimer du SET users.

```
127.0.0.1:6379> SISMEMBER users "Ryan"
(integer) 1
127.0.0.1:6379> SISMEMBER users "Jules"
(integer) 0
127.0.0.1:6379>
```

Cette commande nous permet de rechercher un élément dans un ensemble (SET) grâce à l'opérateur SISMEMBER. Dans ce cas, Ryan est bien présent dans l'ensemble, ce qui fait que la commande nous renvoie 1. En revanche, Jules ayant été supprimé de l'ensemble, la commande nous renvoie 0.

```
127.0.0.1:6379> SADD users2 "Luc"
(integer) 1
127.0.0.1:6379> SADD users2 "Marc"
(integer) 1
127.0.0.1:6379> SUNION users users2
1) "Luc"
2) "Ryan"
3) "Marc"
127.0.0.1:6379>
```

Cette commande nous permet de fusionner deux ensembles (SET) grâce à l'opérateur SUNION. Dans ce cas, nous pouvons constater que les ensembles 'users' et 'users2' ont bien été fusionnés.

Opération sur les ZRANGE :

```
127.0.0.1:6379> ZADD score 10 "Pascal"
(integer) 1
127.0.0.1:6379> ZADD score 15 "Marion"
(integer) 1
127.0.0.1:6379> ZADD score 9 "Chloe"
(integer) 1
```

Cette commande nous permet de créer un SET ORDONNÉ grâce à l'opérateur ZADD. Ici, nous avons créé un SET ORDONNÉ score qui contient le score obtenu de chaque individu.

```
127.0.0.1:6379> ZRANGE score 0 -1
1) "Chloe"
2) "Pascal"
3) "Marion"
127.0.0.1:6379>
```

Cette commande nous permet de récupérer les utilisateurs de notre SET ORDONNÉ rangés dans l'ordre croissant grâce à l'opérateur ZRANGE. Ici, nous pouvons voir que les individus sont rangés par ordre décroissant des scores.

```
127.0.0.1:6379> ZREVRANGE score 0 -1
1) "Marion"
2) "Pascal"
3) "Chloe"
127.0.0.1:6379>
```

Cette commande nous permet de récupérer les utilisateurs de notre SET ORDONNÉ rangés dans l'ordre croissant grâce à l'opérateur ZREVRANGE. Ici, nous pouvons voir que les individus sont rangés par ordre décroissant des scores.

Opération complexe sur les SET ORDONNEE :

```
127.0.0.1:6379> ZRANK score "Marion"
(integer) 2
127.0.0.1:6379>
```

Cette commande nous permet de récupérer la position d'un élément dans notre SET ORDONNÉ grâce à l'opérateur ZRANK. Ici, nous pouvons voir que l'individu Marion est à la position 2.

```
127.0.0.1:6379> ZREM score "Pascal"
(integer) 1
```

Cette commande nous permet de supprimer un élément de notre SET ORDONNÉ grâce à l'opérateur ZREM. Ici, nous supprimons l'individu Pascal avec son score.

```
127.0.0.1:6379> ZRANK score "Marion"
(integer) 1
```

En supprimant Pascal de notre SET ORDONNÉ score, Marion est passée en première position.

Opération sur les HASH :

```
127.0.0.1:6379> HSET users:456 username "Morgan Senechal"
(integer) 1
127.0.0.1:6379> HSET user:456 age 21
(integer) 1
127.0.0.1:6379> HSET user:456 email "morgan.senechal@efrei.net"
(integer) 1
```

Cette commande nous permet de créer un Hash (objet) 'users:456' grâce à l'opération HSET. Ici, nous avons inséré plusieurs informations dans le Hash avec la clé 'users'.

```
127.0.0.1:6379> HGETALL user:456
1) "age"
2) "21"
3) "email"
4) "morgan.senechal@efrei.net"
127.0.0.1:6379>
```

Cette commande nous permet d'afficher toutes les informations présentes dans le Hash associé à une clé grâce à l'opérateur HGETALL. Ici, les différentes informations rattachées à la clé 'user:456' s'affichent.

```
127.0.0.1:6379> HMSET user:123 username "Laurent" age 40 email "laurent@gmail.com"
OK
127.0.0.1:6379> HGETALL user:123
1) "username"
2) "Laurent"
3) "age"
4) "40"
5) "email"
6) "laurent@gmail.com"
127.0.0.1:6379>
```

Cette commande nous permet d'insérer plusieurs valeurs dans notre Hash sur une clé particulière grâce à la commande HMSET. Ici, nous insérons en une seule fois plusieurs informations dans le Hash attaché à la clé '123' que nous affichons ensuite.

```
127.0.0.1:6379> HGET user:123 username
"Laurent"
127.0.0.1:6379>
```

Cette commande nous permet de récupérer des informations précises dans un Hash grâce à la commande HGET. Ici, nous récupérons, dans le Hash 'user:123', l'information attachée à la clé 'username'.

Opération plus complexe :

```
127.0.0.1:6379> HINCRBY user:123 age 10
(integer) 50
127.0.0.1:6379>
```

Cette commande permet d'augmenter un élément d'un Hash grâce à la commande HINCRBY. Ici, nous augmentons l'âge de 'user:123' de 10 qui était à la base défini à 40, mais qui devient maintenant 50.

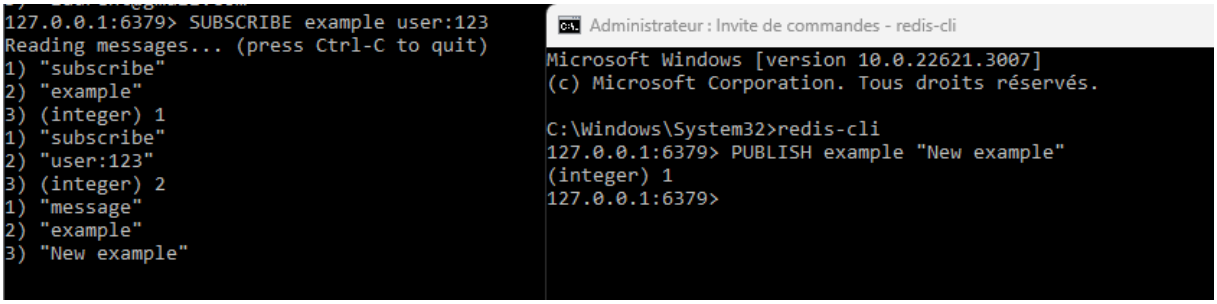
```
127.0.0.1:6379> HKEYS user:123
1) "username"
2) "age"
3) "email"
127.0.0.1:6379>
```

Cette commande nous permet de récupérer toutes les clés d'un Hash grâce à la commande HKEYS. Ici, nous récupérons toutes les clés du Hash 'user:123'.

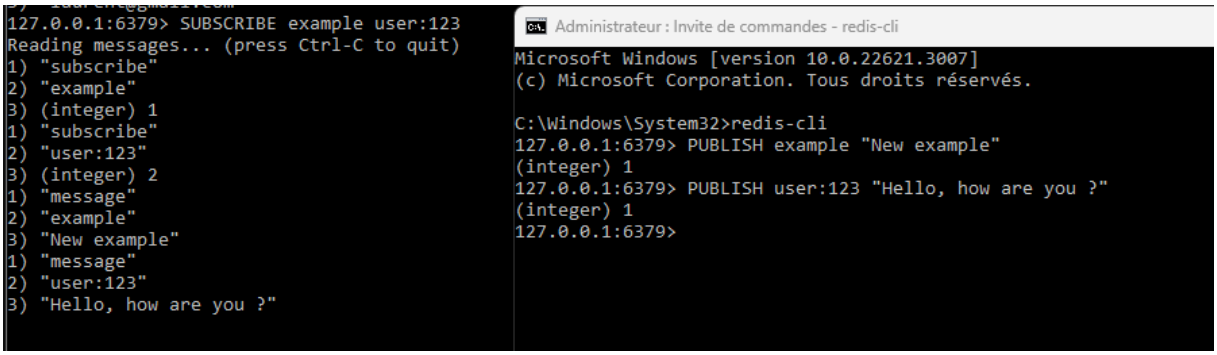
```
127.0.0.1:6379> HVALS user:123
1) "Laurent"
2) "50"
3) "laurent@gmail.com"
127.0.0.1:6379>
```

Cette commande nous permet de récupérer toutes les valeurs d'un Hash grâce à la commande HVALS. Ici, nous récupérons toutes les valeurs du Hash 'user:123'.

Opération sur les Pub/Sub :



Ici, la commande SUBSCRIBE nous permet de souscrire à un canal. Ici, 'user:123' souscrit au canal 'example'. De plus, sur la nouvelle fenêtre de commande (cmd) ouverte, nous pouvons publier un nouveau canal appelé 'New example'. Cela nous permet de faire en sorte que tous les gens qui ont souscrit à 'example' auront accès au canal 'New example'.



De plus, en prenant cet autre exemple avec la commande PUBLISH, nous pouvons envoyer un message à un utilisateur. Ici, nous envoyons le message « Hello, how are you ? » à l'utilisateur 'user:123'.

```
127.0.0.1:6379> PSUBSCRIBE users*
```

Nous pouvons utiliser PSUBSCRIBE pour nous abonner à tous les canaux qui s'abonnent à 'users*'. C'est très pratique pour mettre en place un système à plusieurs niveaux.

Conclusion

Redis se distingue dans l'univers des bases de données en mémoire par sa rapidité, sa flexibilité et sa fiabilité exceptionnelles. Alors que MongoDB et Cassandra ciblent respectivement le stockage de documents et de données réparties, Redis excelle dans les scénarios demandant des performances élevées de lecture et d'écriture. Sa capacité à gérer diverses structures de données et à exécuter des opérations complexes lui confère une valeur inestimable pour les développeurs.

L'extensibilité de Redis avec des modules personnalisables le rend adaptable à une variété de besoins de projets, surpassant des systèmes comme Memcached. Cette adaptabilité, couplée à une intégration aisée dans de nombreux environnements de programmation, fait de Redis un choix de prédilection pour des applications exigeant haute performance et disponibilité.

Redis se positionne donc comme une solution polyvalente et puissante, bien placée pour continuer à jouer un rôle majeur dans l'écosystème des technologies de bases de données.