Delete ll time series : redis-cli FLUSHDB

avoir access a la console : docker exec -it redis-stack redis-cli

Installer redis stack

docker run -p 6379:6379 --name redis-stack redis/redis-stack

redis stack est un conteneur docker qui a plusieurs modules de redist

lancer redis

docker run -p 6379:6379 --name redis-stack redis/redis-stack

liste de modules :

docker exec -it redis-stack redis-cli MODULE LIST

Quelques commandes cles  
TS.CREATE : Crée une nouvelle série temporelle

* TS.ADD : Ajoute un point de données à une série temporelle
* TS.RANGE : Récupère des valeurs sur un intervalle de temps
* TS.MRANGE : Récupère des valeurs de plusieurs séries temporelles
* TS.INFO : Affiche des informations sur une série temporelle

# Création d'une série temporelle pour la température

TS.CREATE sensor:temperature:1 RETENTION 86400000 LABELS sensorId 1 type temperature location "chambre1"

# Création d'une série temporelle pour l'humidité

TS.CREATE sensor:humidity:1 RETENTION 86400000 LABELS sensorId 1 type humidity location "chambre1"

# Ajout de quelques valeurs de test

TS.ADD sensor:temperature:1 \* 22.5

TS.ADD sensor:humidity:1 \* 45

# Récupération des données

TS.RANGE sensor:temperature:1 - +  
  
  
  
lancer le conteneur redis : docker exec -it redis-stack redis-cli

Tester la connexion : PING  
  
# 1. Création de la série principale (données brutes)

TS.CREATE sensor:temperature:1:raw RETENTION 86400000 LABELS sensorId 1 type temperature location "chambre"

# 2. Création d'une série pour stocker les moyennes sur 5 minutes

TS.CREATE sensor:temperature:1:5min RETENTION 86400000 LABELS sensorId 1 type temperature location "chambre" resolution "5min"

# 3. Définition de la règle d'agrégation

TS.CREATERULE sensor:temperature:1:raw sensor:temperature:1:5min AGGREGATION avg 300000

### Instructions détaillées :

**1. Définition du modèle de données**

Pour notre exemple, imaginons que nous surveillons un réseau de capteurs IoT dans un bâtiment avec plusieurs pièces. Chaque capteur mesure :

* Température (°C)
* Humidité (%)
* Qualité de l'air (indice de 0-500)

Script de génération de données   
  
**3. Explication du script**

Ce script fait trois choses importantes :

1. **Crée les séries temporelles** pour différents capteurs dans différentes pièces
2. **Génère des données historiques** sur les 7 derniers jours (pour avoir un jeu de données à analyser)
3. **Génère des données en temps réel** toutes les 30 seconds (simulant des capteurs actifs)

**4. Installation des dépendances et exécution**

Pour utiliser ce script, vous devez installer la bibliothèque Redis pour Python :

pip install redis

Puis exécutez le script :

python generate\_data.py

Le script tournera indéfiniment, générant de nouvelles données toutes les 5 minutes, jusqu'à ce que vous l'arrêtiez avec Ctrl+C.

**5. Vérification des données**

Pour vérifier que les données sont bien insérées, vous pouvez vous connecter à Redis et exécuter :

TS.RANGE sensor:temperature:salon:1 - + COUNT 5

Cela affichera les 5 derniers points de données pour le capteur de température n°1 du salon.

## Étape 4 : Mettre en place RediSearch pour des requêtes avancées

Maintenant que nous avons des données de séries temporelles dans Redis, ajoutons la capacité de recherche avancée avec RediSearch.

### Ce que nous allons faire :

1. Vérifier que RediSearch est disponible
2. Créer un index pour nos séries temporelles
3. Faire des requêtes avancées sur nos données

### Instructions détaillées :

**1. Vérifier la disponibilité de RediSearch**

Si vous utilisez Redis Stack ou une image qui inclut déjà RediSearch, vous n'avez rien à installer. Vérifions simplement qu'il est disponible :

docker exec -it redis-ts redis-cli

MODULE LIST

Vous devriez voir "search" ou "ft" dans la liste des modules.

**2. Créer un index pour nos séries temporelles**

RediSearch nous permet de créer un index sur les labels de nos séries temporelles, ce qui facilite les recherches :

FT.CREATE idx:sensors ON HASH PREFIX 1 sensor: SCHEMA

sensorId TAG SORTABLE

type TAG SORTABLE

location TAG SORTABLE

Cependant, avec RedisTimeSeries, nous utilisons les commandes spécifiques à TimeSeries pour interroger nos données. Les labels que nous avons définis lors de la création des séries temporelles sont déjà indexés et peuvent être utilisés comme filtres.

**3. Exemples de requêtes avancées**

Voici quelques exemples de requêtes que nous pouvons maintenant effectuer :

a) **Obtenir toutes les températures du salon des dernières 24 heures** :

TS.MRANGE - + FILTER location=salon type=temperature

b) **Calculer la température moyenne par pièce sur la dernière heure** :

TS.MRANGE (NOW.ms - 3600000) NOW.ms AGGREGATION avg 3600000 FILTER type=temperature GROUPBY location REDUCE avg

c) **Trouver les 5 moments où l'humidité était la plus élevée dans la salle de bain** :

TS.MRANGE - + FILTER location=salle\_de\_bain type=humidity LIMIT 0 5 AGGREGATION max 3600000

d) **Comparer la qualité de l'air dans toutes les pièces (moyenne sur 24h)** :

TS.MRANGE (NOW.ms - 86400000) NOW.ms AGGREGATION avg 86400000 FILTER type=air\_quality GROUPBY location

**4. Script Python pour analyser les données**

Voici un exemple de script Python qui exécute quelques requêtes et affiche les résultats de manière lisible :

## Étape 5 : Configurer la réplication Redis

La réplication permet de créer des copies de vos données sur plusieurs instances Redis, garantissant ainsi la disponibilité même en cas de défaillance d'un serveur.

### Ce que nous allons faire :

1. Créer une architecture maître-esclave (primaire-réplica)
2. Configurer plusieurs instances Redis
3. Vérifier que la réplication fonctionne correctement

### Instructions détaillées :

**1. Configuration du maître (primaire) et des esclaves (réplicas)**

Pour une configuration de base, nous aurons besoin de lancer au moins 3 instances Redis :

* 1 instance maître (primaire)
* 2 instances esclaves (réplicas)

**2. Création des conteneurs Redis**

Commençons par créer le maître :

# Arrêtez d'abord votre instance Redis actuelle si elle tourne

docker stop redis-ts

Cependant, il y a un problème : les conteneurs Docker ne peuvent pas se voir par défaut avec les noms d'hôte. Nous devons créer un réseau Docker :

# Créer un réseau Docker

docker network create redis-net

# Recréer les conteneurs sur le réseau

docker run -d --name redis-master --network redis-net -p 6379:6379 redis/redis-stack

docker run -d --name redis-replica1 --network redis-net -p 6380:6379 redis/redis-stack redis-server --replicaof redis-master 6379 --loadmodule /opt/redis-stack/lib/redisearch.so --loadmodule /opt/redis-stack/lib/redistimeseries.so --loadmodule /opt/redis-stack/lib/rejson.so

docker run -d --name redis-replica2 --network redis-net -p 6381:6379 redis/redis-stack redis-server --replicaof redis-master 6379 --loadmodule /opt/redis-stack/lib/redisearch.so --loadmodule /opt/redis-stack/lib/redistimeseries.so --loadmodule /opt/redis-stack/lib/rejson.so

**3. Vérification de la réplication**

Vérifions que la réplication fonctionne correctement. Connectez-vous d'abord au maître :

docker exec -it redis-master redis-cli

Exécutez la commande INFO pour vérifier les réplicas connectés :

INFO replication

Vous devriez voir quelque chose comme :

# Replication

role:master

connected\_slaves:2

slave0:ip=172.18.0.3,port=6379,state=online,offset=42,lag=0

slave1:ip=172.18.0.4,port=6379,state=online,offset=42,lag=0

**4. Test de la réplication**

Pour tester la réplication :

1. Créez une série temporelle sur le maître :

TS.CREATE test:replication RETENTION 86400000 LABELS type test location master

TS.ADD test:replication \* 42.0

1. Vérifiez que les données sont répliquées sur les esclaves :

# Quittez le client redis-cli avec 'exit'

# Connectez-vous au premier réplica

docker exec -it redis-replica1 redis-cli

INFO replication

TS.RANGE test:replication - +

Vous devriez voir les données dans le réplica. Notez que les réplicas sont en lecture seule - vous ne pouvez pas y écrire de données.

Cette configuration de base de réplication offre plusieurs avantages :

* Haute disponibilité des données
* Possibilité de distribuer les requêtes de lecture
* Récupération en cas de défaillance du maître

Maintenant que la réplication est configurée, passons à la fragmentation (sharding) des données.

## Nouvelle étape : Mettre en place RediSearch avec Redis Stack

### Ce que nous allons faire :

1. Installer Redis Stack avec tous les modules
2. Configurer RediSearch pour nos données de séries temporelles
3. Créer des index et effectuer des requêtes avancées

### Instructions détaillées :

**1. Installation de Redis Stack**

Commençons par arrêter vos conteneurs Redis actuels et installer Redis Stack :

# Arrêter les conteneurs existants

docker stop redis-master redis-replica1 redis-replica2

# Lancer Redis Stack

docker run -d --name redis-stack -p 6379:6379 redis/redis-stack

**2. Vérification des modules installés**

Vérifions que tous les modules sont bien présents :

docker exec -it redis-stack redis-cli MODULE LIST

Vous devriez maintenant voir plusieurs modules, dont "timeseries" et "search" (ou "ft").

**3. Génération de données de test**

Pour générer rapidement des données de test, vous pouvez utiliser le script Python que nous avons créé précédemment. Si nécessaire, modifiez le script pour qu'il se connecte à votre nouvelle instance Redis Stack.

**4. Création d'index RediSearch**

RediSearch fonctionne principalement avec des structures de type HASH dans Redis. Cependant, nous pouvons utiliser RediSearch pour créer des index qui facilitent la recherche dans les métadonnées (labels) de nos séries temporelles.

Connectez-vous à Redis et créez un index :

docker exec -it redis-stack redis-cli

Puis créez un index sur les labels de séries temporelles :

FT.CREATE idx:sensors ON HASH PREFIX 1 ts:sensor: SCHEMA

sensorId TAG SORTABLE

type TAG SORTABLE

location TAG SORTABLE

**5. Script Python pour créer l'index et alimenter des données**

Script de configuration pour Redis Stack avec RediSearch

Code

Ce script réalise plusieurs choses importantes :

1. **Crée un index RediSearch** pour nos capteurs
2. **Configure des séries temporelles** pour chaque capteur
3. **Crée des structures HASH** pour les métadonnées (nécessaires pour RediSearch)
4. **Génère des données fictives** sur 3 jours
5. **Exécute des requêtes de recherche avancées**

**6. Exécution des requêtes de recherche avancées**

Une fois le script exécuté, vous pouvez utiliser des requêtes RediSearch directement depuis le CLI Redis :

# Tous les capteurs de température

FT.SEARCH idx:sensors "@type:{temperature}"

# Capteurs du salon avec humidité > 50%

FT.SEARCH idx:sensors "@location:{salon} @type:{humidity} @lastValue:[50 +inf]"

# Les 3 capteurs avec la plus mauvaise qualité d'air

FT.SEARCH idx:sensors "@type:{air\_quality}" SORTBY lastValue DESC LIMIT 0 3

**7. Avantages de RediSearch**

RediSearch offre plusieurs avantages pour notre projet :

* Recherche complexe sur les métadonnées des capteurs
* Filtrage par plages de valeurs
* Tri et pagination des résultats
* Agrégations et facettes pour les analyses

Maintenant que nous avons configuré RediSearch, nous pouvons passer à l'étape suivante : la mise en place de la réplication et de la fragmentation pour la haute disponibilité et la scalabilité de notre système.

Souhaitez-vous continuer avec la mise en place de la réplication comme prévu précédemment ?

Retry

A

d'accord je comprend nous pouvons continuer

Edit

Parfait ! Continuons avec la mise en place de la réplication et de la fragmentation pour Redis.

## Étape 6 : Configuration de la réplication et de la fragmentation

### Ce que nous allons faire :

1. Configurer la réplication maître-esclave avec Redis Stack
2. Mettre en place la fragmentation (sharding) des données
3. Combiner réplication et fragmentation

### Instructions détaillées :

**1. Mise en place du réseau Docker**

Pour que nos conteneurs Redis puissent communiquer entre eux, commençons par créer un réseau Docker dédié :

docker network create redis-net

**2. Configuration de la réplication**

Nous allons créer un cluster Redis avec 2 groupes de réplication, chacun comprenant 1 maître et 1 esclave :

# Arrêter les conteneurs existants

docker stop redis-stack

docker rm redis-stack

# Créer le premier groupe (Shard 1)

docker run -d --name redis-master1 --network redis-net -p 6379:6379 redis/redis-stack

docker run -d --name redis-replica1 --network redis-net -p 6380:6379 redis/redis-stack redis-server --replicaof redis-master1 6379

# Créer le deuxième groupe (Shard 2)

docker run -d --name redis-master2 --network redis-net -p 6381:6379 redis/redis-stack

docker run -d --name redis-replica2 --network redis-net -p 6382:6379 redis/redis-stack redis-server --replicaof redis-master2 6379

**3. Vérification de la réplication**

Vérifions que la réplication fonctionne pour chaque groupe :

# Vérifier le premier groupe

docker exec -it redis-master1 redis-cli INFO replication

# Vérifier le deuxième groupe

docker exec -it redis-master2 redis-cli INFO replication

**4. Mise en place de la fragmentation**

Pour la fragmentation, nous devons décider comment distribuer nos données. Une approche simple consiste à utiliser une stratégie basée sur la localisation des capteurs :

* Shard 1 (redis-master1) : capteurs des zones "salon", "cuisine"
* Shard 2 (redis-master2) : capteurs des zones "chambre1", "chambre2", "salle\_de\_bain"

**5. Script Python pour implémenter la fragmentation**

Script pour la réplication et fragmentation Redis

Code

Ce script est conçu pour travailler avec notre architecture d'instances Redis répliquées et fragmentées. Il fait plusieurs choses importantes :

1. **Définit une stratégie de fragmentation** basée sur la localisation
2. **Se connecte à tous les shards** (maîtres et réplicas)
3. **Crée des index RediSearch** sur chaque shard
4. **Distribue les données** selon la stratégie de fragmentation
5. **Vérifie que la réplication fonctionne** correctement
6. **Exécute des requêtes sur tous les shards** et combine les résultats

Pour utiliser ce script, vous devez ajuster les paramètres de connexion selon votre configuration réseau.

**6. Architecture combinée : réplication + fragmentation**

Cette architecture offre plusieurs avantages :

* **Haute disponibilité** : si un maître tombe en panne, son réplica peut prendre le relais
* **Scalabilité** : vous pouvez ajouter plus de shards pour gérer davantage de données
* **Équilibrage de charge** : les lectures peuvent être dirigées vers les réplicas
* **Isolation des pannes** : un problème sur un shard n'affecte pas les autres

Votre système peut maintenant fonctionner sur plusieurs machines, avec des données répliquées et fragmentées pour assurer à la fois sécurité et performance.