SGBLE

Geste de savoir

Backup d'une base Redis

24 mai 2021

Table des matières

Ce bref billet présente une façon de faire un backup d'un *snapshot* de Redis, aussi dit un *dump* (en mode RDB). Cela s'est avéré moins trivial que ce que j'imaginais. Voyons-voir.

1. SAVE ou BGSAVE

Redis fait lui-même un snapshot régulier selon des paramètres à définir dans sa configuration (voire aucun snapshot si configuré comme tel). Le dump qu'il laisse se situe souvent dans `/var/lib/redis/dump.rdb`. Pour faire un backup (copie du dump), c'est donc assez simple: simplement copier `/var/lib/redis/dump.rdb` quelque part et le tour est joué. Mais je préfère bien entendu avoir une version fraîche du dump avant de procéder au backup, pas celle qui date de la dernière mise à jour par Redis.

Redis propose deux commandes pour forcer «à la main» le snapshot d'un dump RDB: `SAVE` et `BGSAVE`. On évitera `SAVE` car la commande est synchrone et bloque ainsi l'intégralité des activités de la base de données jusqu'à la fin de l'opération. Cependant, `BGSAVE` ne souffre pas du même problème et est donc préconisé: la commande est asynchrone et retourne la main immédiatement et évite de bloquer l'activité de la base de données.

?

> Mais si l'opération est asynchrone, comment savoir quand l'opération prendra fin afin de procéder à la copie du dump frais?

C'est là qu'il faudra ruser. On peut éventuellement se baser sur la date de dernière modification du fichier `/var/lib/redis/dump.rdb` et attendre qu'il change avant de lancer la copie. Mais il y a une méthode plus sophistiquée et fiable grâce à l'appel système `inotify` de Linux.

2. inotifywait

On peut analyser le comportement de Redis grâce à l'appel système `inotify` sous Linux. Quand le noyau détecte un événement concernant un certain fichier, il nous alerte. J'utilise la commande `inotifywait` pour cela.

Première étape, on «arme» un moniteur sur le fichier pour l'observer:

```
# inotifywait -m /var/lib/redis/dump.rdb
```

Seconde étape: on exécute un `BGSAVE` dans Redis (dans un autre terminal):

```
% redis-cli
127.0.0.1:6379> BGSAVE
Background saving started
```

Enfin, on revient à notre moniteur et on observe cela:

```
# inotifywait -m /var/lib/redis/dump.rdb
Setting up watches.
Watches established.
/var/lib/redis/dump.rdb ATTRIB
/var/lib/redis/dump.rdb DELETE_SELF
```

L'événement `DELETE_SELF` signe la fin de l'opération d'un `BGSAVE`. Cet événement signifie que le dump a été supprimé. En effet, Redis ne modifie donc pas *in situ* le fichier `/var/lib/redis/dump.rdb`, mais il se contente de le remplacer par un nouveau dump, ce qui se traduit par une suppression de ce dernier en vue du remplacement.

Voici le processus complet commenté:

```
/var/lib/redis/ CREATE temp-17672.rdb  # Redis créé un dump
    temporaire "temp-17672.rdb"
/var/lib/redis/ OPEN temp-17672.rdb
/var/lib/redis/ MODIFY temp-17672.rdb  # Il écrit dedans ses
    données
/var/lib/redis/ CLOSE_WRITE,CLOSE temp-17672.rdb
/var/lib/redis/ MOVED_FROM temp-17672.rdb  # Quand le dump est
    écrit, il écrase "dump.rdb"
/var/lib/redis/ MOVED_TO dump.rdb
```

3. Script de backup d'un snapshot

On connaît un élément sur lequel se baser. Très bien, il suffit maintenant de faire un script qui reprend le même principe: armer un moniteur `inotify`, attendre l'événement `DELETE_SELF`, puis procéder à la copie du dump frais aussitôt.

J'ai eu besoin de faire deux processus concurrents pour cela. De la même façon que précédemment j'ai eu à lancer en parallèle l'exécution du moniteur et de la commande Redis dans deux terminaux.

En Bash, c'est heureusement assez simple de faire cela. Un bloc (entre parenthèses) représente un processus qui contient une ou plusieurs commandes. Le `&` permet de ne pas attendre la fin de l'exécution d'un processus avant de lancer le suivant.

Je vous invite à faire le test dans votre terminal. Comparez `(sleep 10; echo "fini"); (echo "Instantané")` avec `(sleep 10; echo "fini")& (echo "Instantané"); wait`. Dans le premier cas, on doit attendre 10 seconde avant d'afficher «Instantané». Dans le second cas, le second processus s'exécute aussitôt sans attendre que le premier s'achève, affichant ainsi «Instantané». Grâce au `wait`, il attend toutefois que le premier processus se termine avec un «fini» avant de rendre la main.

Voici ce que cela donne dans un script:

```
#!/bin/bash

dump_location=/var/lib/redis/dump.rdb
destination=/tmp/dump-$(date +"%Y%m%d").rdb

(
    if inotifywait -e DELETE_SELF "$dump_location"; then
        cp "$dump_location" "$destination" &&
        echo OK
    else
        echo "no event has occurred"
        exit 1
    fi

)& (
    sleep 1
    echo "BGSAVE" | redis-cli
    echo "command sent"
)

wait
```

Dans le second processus, on attend une bonne seconde histoire de laisser le temps à `inotifywait -e DELETE_SELF "$dump_location"` de s'armer et d'être prêt à écouter (ce qui est probablement fait en quelques microsecondes max…). Il n'y a plus qu'à lancer le `BGSAVE` qui ordonne à Redis de commencer le *snapshotting* et rend aussitôt la main, ce qui nous fait donc sortir de ce processus.

Mais grâce au `wait`, on attend que *tous* les processus du script soient finis. Ainsi, même si le dump final tarde à arriver dans le premier processus (ce qui peut être le cas si le dump est de quelques gigaoctets), on s'assure de bien l'attendre et de procéder à sa copie avant que le script ne s'achève pour de bon.

Si l'événement précis `DELETE_SELF` n'est pas remonté, alors la commande échoue avec le message `"no event has occurred"` dans la branche du `else`. Donc si jamais une opération sur `dump.rdb` inattendue et indépendante du script se présentait, cela mettrait aussitôt fin au script avec une erreur.

? Et si c'est un évènement `DELETE_SELF` *indépendant* du script qui se présentait?

Dans ce cas, le script procède quand même à la copie du dump, même si son apparition n'est pas issue de son propre fait. La commande `BGSAVE` lancée par le script n'aura donc aucun effet subséquent car le moniteur `inotify` n'est armé qu'un seule fois. Il n'y aurait donc qu'une seule copie du dump.

Ce cas peut parfaitement arriver, notamment quand Redis décide tout seul de faire le snapshot. Si ça tombe au même moment que l'exécution du script (dans une fenêtre de ~1 seconde), le cas de figure se présenterait.

Pour moi, ce n'est pas un souci car tout ce qui m'intéresse c'est d'avoir un dump frais et à jour. Même si je ne prends pas en compte le dump provoqué par *mon* `BGSAVE` ce n'est pas fondamentalement grave. J'ai donc choisi de ne pas complexifier le script et de le laisser faire.

Dans mon cas, je copie le dump dans `/tmp` pour le manipuler plus facilement par la suite. Mais il est bien entendu possible de faire autre chose, par exemple enregistrer son snapshot directement dans un Cloud AWS, par exemple:

```
1  BUCKET=mon_joli_bucket
2  destination_file=dump-$(date +"%Y%m%d").rdb
3
4  (
5      if inotifywait -e DELETE_SELF "$dump_location"; then
6          aws s3 cp "$dump_location" s3://$BUCKET/$destination_file
               &&
7          echo OK
8  ...
9  ...
10 ...
```

Dans la plupart des cas, une base Redis n'a pas spécialement vocation à maintenir des données de façon pérenne (bien qu'il soit capable de le faire selon la façon dont on le configure). Le principe d'un snapshotting aussi fin peut donc paraître curieux. Cependant, le script a au moins l'avantage de pouvoir gérer le cas où un snapshot ponctuel de Redis apparaîtrait, plutôt que de simplement copier à l'aveuglette le fichier `/var/lib/redis/dump.rdb` sans se préoccuper de savoir s'il sera remplacé pendant la copie.