Engineer

Redis

Stockage en mémoire pour des données de courte durée

Ce que c’est

Redis est un serveur de bases de données qui conserve tout son jeu de données en mémoire sous forme de structures typées et permet à de nombreux processus applicatifs de les lire et de les modifier par le réseau, une commande courte à la fois. Le projet se décrit comme un serveur de structures de données. L’édition disponible gratuitement s’appelle Redis Open Source, et Redis Ltd vend aussi Redis Cloud et Redis Software sous licences commerciales.

Chaque clé contient une valeur d’un seul type. Les chaînes sont de simples octets et servent aussi de compteurs, les tables de hachage sont des enregistrements de champs et de valeurs, les listes conservent l’ordre d’arrivée des éléments, les ensembles contiennent des membres uniques et les ensembles triés classent leurs membres selon un score. La liste des types se prolonge avec les flux, les documents JSON, les index géospatiaux, les séries temporelles, les ensembles de vecteurs et des structures probabilistes comme les filtres de Bloom et HyperLogLog.

Les commandes agissent directement sur ces types. `INCR` incrémente un compteur d’une unité et renvoie sa nouvelle valeur, `SADD` ajoute des membres à un ensemble et `SPOP` en retire un. Toute clé peut avoir un délai au terme duquel Redis la supprime lui-même. Redis qualifie une telle clé de volatile. Lorsque plusieurs étapes doivent se dérouler comme une seule, un script Lua s’exécute dans le serveur et Redis garantit que rien d’autre ne s’exécute pendant ce temps.

Les données vivent en mémoire et le disque est facultatif. Redis peut écrire des instantanés à un instant donné, les fichiers RDB, ou journaliser chaque écriture dans un fichier en ajout seulement, AOF, qu’il rejoue au démarrage, ou faire les deux, ou aucun des deux. Les répliques copient un serveur primaire et peuvent

servir les lectures. Un plafond mémoire, `maxmemory`, et une politique d'éviction déterminent ce qui se passe quand les données dépassent l'espace qui leur est alloué.

La licence a changé avec Redis 8. Depuis cette version, Redis Open Source est proposé au choix sous trois licences : RSALv2, SSPLv1 et AGPLv3. Les deux premières donnent accès au code source ; AGPLv3 est l'option open source.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que Redis ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Identifier les requêtes et les index manquants dans un système alourdi au fil des ans, puis corriger ce qui compte.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. 30 jours de données électriques traités en 180 ms — 180 ms pour agréger un mois de relevés de compteurs

30 jours de données électriques traités en 180 ms Contactez-nous →

Pourquoi c'est important

Une application web s'exécute rarement dans un seul processus. Derrière un répartiteur de charge se trouvent plusieurs instances, chacune avec sa propre mémoire. Tout ce qui y est conservé — nombre de tentatives de connexion, session, page en cache — n'existe que dans le processus qui a traité la requête. Il suffit d'arriver sur une autre instance pour contourner une limite comptée ainsi. Redis donne à toutes les instances un emplacement commun pour ce type d'état, assez rapide pour être consulté à chaque requête.

Les commandes atomiques comptent autant que la vitesse. Deux instances qui lisent chacune un compteur, ajoutent un et réécrivent peuvent toutes deux écrire le même nombre : une incrémentation disparaît. `INCR` effectue la lecture et l'écriture dans le serveur en une seule étape, si bien que cent requêtes simultanées augmentent le compteur d'exactement cent. Un script Lua étend cette garantie à une courte séquence : une vérification et une modification qui ne doivent pas être séparées, par exemple vérifier qu'il reste une place et la prendre, s'exécutent d'un seul bloc.

L'expiration est le troisième élément. Une fenêtre de limitation de débit, une session ou un code à usage unique devrait disparaître une fois son délai écoulé. Avec un délai sur la clé, le nettoyage revient au stockage plutôt qu'à une tâche planifiée.

Le prix est la durabilité, et mieux vaut connaître les chiffres avant de choisir. Avec le fichier en ajout seulement et sa politique par défaut d'un fsync par seconde, un plantage peut coûter environ une seconde d'écritures. Avec les seuls instantanés, généralement pris toutes les quelques minutes, la documentation indique que vous « devriez être prêt à perdre les dernières minutes de données ». La réplication est asynchrone par défaut : un primaire peut donc confirmer une écriture et s'arrêter avant qu'une réplique l'ait reçue. Redis convient ainsi aux états qui peuvent être reconstruits ou perdus sans dommage, mais ne convient guère à l'unique copie d'une commande, d'un paiement ou d'un solde. La question de conception essentielle est ce que fait l'application quand Redis est inaccessible ou a oublié quelque chose qui lui avait été confié.

À quoi cela ressemble en pratique

22 limiteurs de débit conservent leur état dans Redis sur un annuaire public d'entreprises et de professionnels. La limite tient ainsi sur toutes les instances au lieu de vivre dans un seul processus, où elle serait triviale à contourner. Chaque limiteur est conçu pour le chemin qu'il protège, car une tentative de connexion, une recherche et un export massif deviennent abusifs à des fréquences très différentes. Le choix déterminant est ce qui se passe en l'absence de Redis : les limiteurs ferment l'accès en cas de panne et le trafic est rejeté au lieu de passer. Ce choix a un coût visible. Un problème Redis devient un problème perceptible par les utilisateurs, accepté délibérément parce que l'autre possibilité serait qu'il devienne un problème de facturation.

Pour vendre un stock limité, Redis sert de filtre rapide devant une base de données : un compteur Redis décrémenté par un script Lua admet approximativement autant d'acheteurs qu'il y a d'unités et écarte les autres en quelques microsecondes. Le script réunit vérification et décrémentation en une seule étape. Ce filtre promet moins qu'il n'y paraît. Redis réplique de manière asynchrone : après un basculement, le nouveau primaire peut avoir oublié certaines admissions et laisser passer quelques acheteurs de trop. Un `UPDATE` conditionnel dans la base décide toujours de chaque vente, et un processus de réconciliation réinitialise le filtre d'après le nombre indiqué par la base. Quand les unités sont peu nombreuses et distinctes, comme des places numérotées, leurs identifiants peuvent être placés dans un ensemble Redis et `SPOP` les distribue un par un, sans concurrence sur un compteur.

Le rôle le plus ordinaire est celui d'un simple cache. Une infrastructure Azure définie en Bicep construit chaque environnement à partir des mêmes modèles, qu'il s'agisse de test, de préproduction ou de production. Chacun dispose de

Redis pour le cache, à côté d'un PostgreSQL Flexible Server, des Container Apps qui exécutent l'API et le site web, et de Front Door avec une politique WAF. Le cache étant déclaré dans les modèles avec tout le reste, le modifier passe par une pull request comme toute autre modification d'environnement.

Où les problèmes surviennent

La plupart des problèmes Redis viennent de l'assimilation d'un stockage rapide mais oublieux à un stockage durable, ou de l'hypothèse qu'il se souvient de ce qui lui a été indiqué sur le temps.

Une clé censée expirer reste indéfiniment. `SET` remplace la valeur et supprime tout délai d'expiration qu'avait la clé. Un code qui actualise une valeur en cache avec un simple `SET` transforme donc une entrée de dix minutes en entrée permanente :

```
> SET session:42 "v1" EX 600
OK
> SET session:42 "v2"
OK
> TTL session:42
(integer) -1
```

Passez le délai à chaque écriture qui en a besoin ou ajoutez `KEEPTTL`, disponible depuis 6.0.0, pour conserver l'ancien. La même fuite se cache dans un compteur construit avec `INCR` suivi d'un `EXPIRE` distinct : si la seconde commande n'arrive jamais, la clé n'expire jamais. Envoyez les deux dans un seul bloc `MULTI` et `EXEC`, ou dans un seul script Lua.

Les écritures renvoient des erreurs alors que les lectures fonctionnent encore. Les données ont atteint `maxmemory` et la politique d'éviction est `noeviction`, ou une politique `volatile-` appliquée à un jeu de données où aucune clé n'a de délai, ce qui revient au même. Choisissez la politique selon le besoin : une politique `allkeys-` pour un cache pur, où toute clé peut disparaître, et une politique `volatile-` uniquement si les clés en cache partagent une instance avec des clés devant rester. Pour ce mélange, la documentation conseille d'envisager deux instances distinctes.

Des écritures confirmées par Redis manquent après un redémarrage ou un basculement. Un plantage est survenu dans la fenêtre fsync, ou une réplique a été promue avant d'avoir reçu l'écriture. C'est le fonctionnement prévu de Redis : la correction porte donc sur ce qui y est stocké. Conservez tout ce qui doit survivre dans une base aux commits durables, placez Redis devant comme filtre ou cache et réconciliez les deux depuis la base, comme le fait la note sur le stock.

Tous les clients se bloquent au même moment. Un long script Lua s'exécute, et Redis bloque toute autre activité du serveur pendant toute sa durée. Gardez des scripts courts et bornés, sans boucles sur de grandes collections.

Un limiteur laisse passer le trafic pendant une panne. Le code client interprète une erreur Redis comme « limite non atteinte » : le limiteur s'ouvre précisément quand son stockage est inaccessible. Choisissez volontairement le comportement en cas de panne et testez-le en arrêtant Redis. Un limiteur protégeant un chemin coûteux devrait refuser, comme les 22 limiteurs ci-dessus, avec pour coût de rendre le problème Redis visible aux utilisateurs.

Pour aller plus loin

- Développer avec Redis — le point de départ du projet : commandes à essayer, bibliothèques clientes par langage et types de données.
- Persistance Redis — instantanés, fichier en ajout seulement, politiques fsync et pertes possibles pour chaque option.
- Réplication Redis — suivi d'un primaire par les répliques et raison pour laquelle une fenêtre de perte de données subsiste.
- Éviction des clés — `maxmemory` , toutes les politiques d'éviction et conseils pour en choisir une.
- EXPIRE — délais des clés et commandes qui les suppriment ou les conservent.
- Scripts Lua — exécution atomique d'un script sur le serveur et activité bloquée pendant ce temps.

Glossaire

Choisissez selon vos contraintes réelles

Présentez un jeu de données représentatif, les questions auxquelles il doit répondre et le goulot d'étranglement actuel. Nous pouvons convenir de la manière de mesurer la qualité, le débit et la reprise.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce que Redis ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/redis/