Report stage 5

REDIS

Author: Nguyen Trung Hieu

Outline

I. Tổng quan kiến thức học được

- Cài đặt server redis
- Tạo 1 cụm Redis và tạo replica cho server redis
- Các kiểu dữ liệu của Redis
- Publish/ Subscribe
- Stream

Một số ứng dụng của Redis

CACHING

Redis là một lựa chọn tuyệt vời để triển khai in-memory cache để giảm độ trễ truy cập dữ liệu, tăng thông lượng và giảm tải khỏi cơ sở dữ liệu quan hệ trong chương trình của bạn. Redis có thể phục vụ các dữ liệu được yêu cầu thường xuyên ở thời gian phản hồi dưới một phần nghìn giây. Và cho phép bạn dễ dàng mở rộng quy mô cho các tải cao hơn mà không cần tăng phụ trợ tốn kém. Persistent session caching, web page caching và caching của các đối tượng được sử dụng thường xuyên như hình ảnh, tệp và siêu dữ liệu là các ví dụ phổ biến về caching với Redis.

SESSION STORE

Redis như một kho lưu trữ dữ liệu trong bộ nhớ với tính sẵn sàng cao và bền bỉ là lựa chọn phổ biến của các nhà phát triển ứng dụng để lưu trữ và quản lý dữ liệu session cho các ứng dụng của mình. Redis cung cấp độ trễ dưới một phần nghìn giây và khả năng phục hồi cần thiết để quản lý dữ liệu session như hồ sơ người dùng, thông tin đăng nhập, trạng thái phiên và cá nhân hóa cụ thể của người dùng.

REAL-TIME ANALYTICS

Redis có thể được sử dụng với các giải pháp streaming như Apache Kafka và Amazon Kinesis như một in-memory data store để nhập, xử lý và phân tích dữ liệu real-time với độ trễ nhỏ hơn một phần nghìn giây. Redis là một lựa chọn lý tưởng cho các hệ thống real-time analytics như truyền thông xã hội, quảng cáo và IoT.

II. Cài đặt server redis

Hiện tại Redis không hỗ trợ cho Window, nhưng có thể cài đặt Redis thông qua WSL và cài đặt tương tự như Linux

A. Cài đặt trên Linux

You can install recent stable versions of Redis from the official **packages.redis.io** APT repository.

> **Prerequisites**
>
> If you're running a very minimal distribution (such as a Docker container) you may need to install **lsb-release**, **curl** and **gpg** first:
>
> ```
> sudo apt install lsb-release curl gpg
> ```

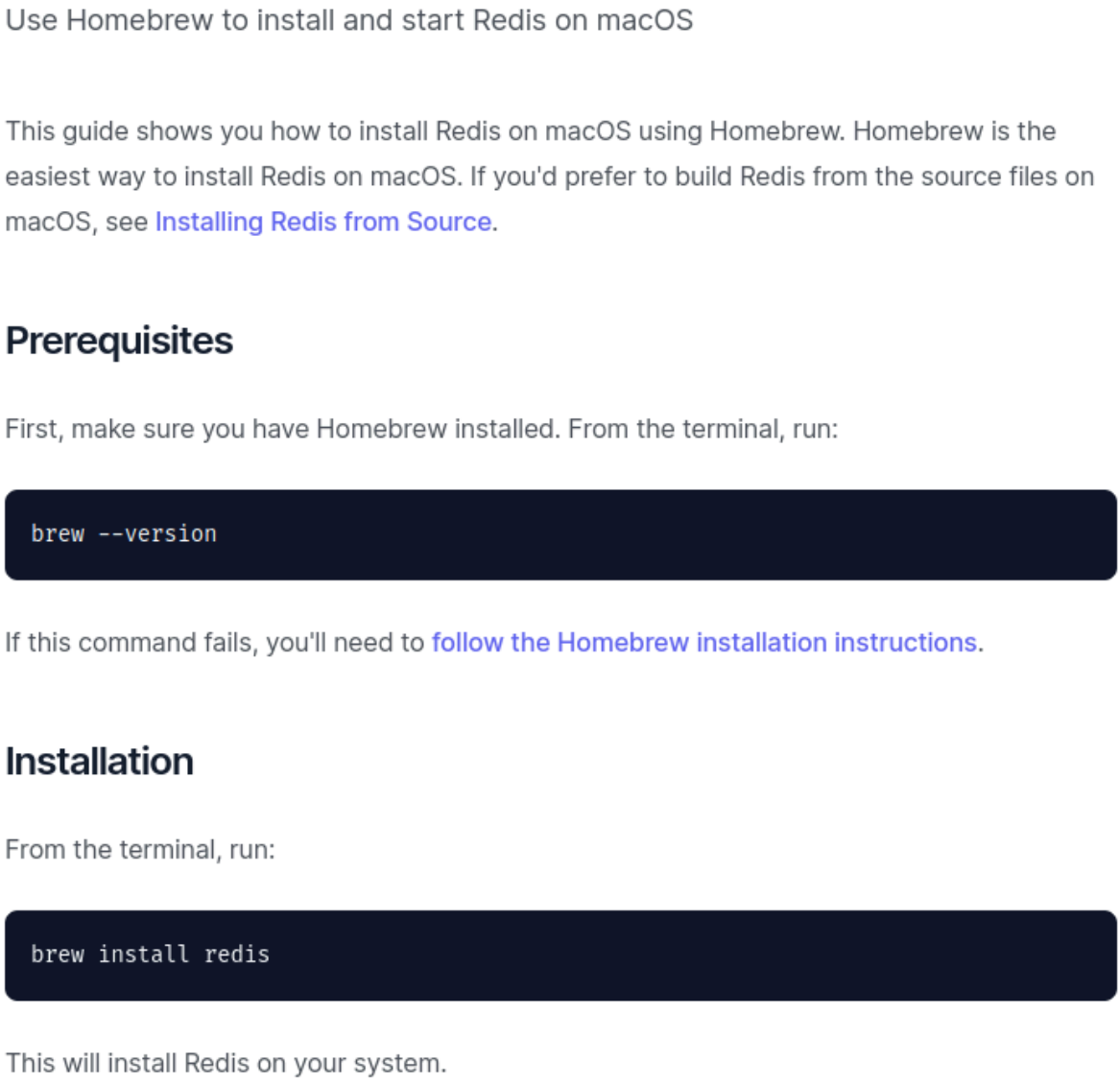
Add the repository to the **apt** index, update it, and then install:

```
curl -fsSL https://packages.redis.io/gpg | sudo gpg --dearmor -o /usr/share/keyrings/redis

echo "deb [signed-by=/usr/share/keyrings/redis-archive-keyring.gpg] https://packages.redis

sudo apt-get update
sudo apt-get install redis
```

B. Cài đặt trên MacOs



Dưới góc nhìn của bản thân, nhìn chung redis rất dễ dàng cài đặt và sử dụng do Redis cung cấp command redis-cli và redis-server dễ dàng tương tác với server, không chỉ vậy doc của redis rất đầy đủ và chi tiết

III. Tạo và sử dụng 1 cụm Redis

Trước hết ta cần config Redis server cho phép chế độ cluster tại mỗi nút.

Mở file redis.cof và config cơ bản như sau:

```
port 7000
cluster-enabled yes
cluster-config-file nodes.conf
cluster-node-timeout 5000
appendonly yes
```

Chúng ta có thể config nhiều thuộc tính hơn để phù hợp với yêu cầu của hệ thống

Sau đó, ta sẽ khởi động từng node với file config như trên:

```
redis-server ./redis.conf
```

Cuối cùng ta chạy lệnh redis-cli để tạo thành cụm:

```
redis-cli --cluster create 127.0.0.1:7000 127.0.0.1:7001 \
127.0.0.1:7002 127.0.0.1:7003 127.0.0.1:7004 127.0.0.1:7005 \
--cluster-replicas 1
```

Với câu lệnh trên thì mỗi node chính sẽ có một node là replica do có tham số –cluster-replicas 1 (có thể đạt nhiều hơn). Nếu số lượng node không đủ để tạo replica thì sẽ tồn tại lớp chính không có replica

IV. Các kiểu dữ liệu của redis

Redis lưu trữ dữ liệu theo key-value , key và value có kiểu int, embstr, raw string
Ngoài ra Redis còn hỗ trợ các cấu trúc dữ liệu khác như List, Set, SortedSet, Hash, Hyperloglog
Với mỗi cấu trúc dữ liệu cung cấp các phương thức tương tác riêng

1. **Strings**
   rong Redis, chuỗi được biết đến là 'an toàn nhị phân'. Một chuỗi Redis đại diện cho bất kỳ dữ liệu nào, bất kể loại của nó. Điều này có thể là dữ liệu văn bản, như tên hoặc email của người dùng, hoặc dữ liệu nhị phân như một hình ảnh hoặc một đối tượng được chuỗi hóa. Các giá trị chuỗi thực sự có thể lưu trữ các giá trị của các loại khác nhau, điều này khiến cho kiểu dữ liệu chuỗi trở nên khá linh hoạt.

Một chuỗi Redis có thể lưu trữ lên đến 512 MB dữ liệu, và giới hạn tối đa này áp dụng độc lập cho cả khóa và giá trị tương ứng của nó. Do các hạn chế về kích thước này và tính chất lưu trữ trong bộ nhớ của Redis, các hoạt động trên chuỗi thường nhanh chóng.

- Lệnh SET được sử dụng để gán một khóa với một giá trị chuỗi. Nếu khóa đã tồn tại, lệnh này sẽ ghi đè lên giá trị cũ.

  ```
  redis-cli SET username “john_doe”
  ```

  Ở đây, khóa "username" được đặt để có giá trị "john_doe".

- Lệnh GET lấy giá trị được liên kết với một khóa cụ thể.

  ```
  redis-cli GET username
  ```

  Đối với ví dụ trước đó của chúng ta, điều này sẽ trả về giá trị "john_doe" được liên kết với khóa "username".

- Lệnh DEL xóa giá trị được liên kết với một khóa.

  ```
  redis-cli DEL username
  ```

  Điều này sẽ xóa giá trị "john_doe" đã được đặt.

2. **List**

Danh sách Redis là các bộ sưu tập được sắp xếp của chuỗi. Chúng là duy nhất vì chúng được thực hiện dưới dạng các danh sách liên kết. Điều này có nghĩa là các hoạt động ở đầu hoặc cuối danh sách, như thêm hoặc xóa các phần tử, được thực hiện trong thời gian hằng số, làm cho chúng cực kỳ nhanh chóng. Kiểu dữ liệu này đặc biệt hữu ích khi bạn muốn duy trì một thứ tự của các phần tử như một dòng thời gian các hoạt động hoặc một nhật ký các sự kiện.
1. Lệnh LPUSH chèn một giá trị mới vào đầu danh sách. Nếu khóa không tồn tại, một danh sách mới sẽ được tạo ra.

```
redis-cli LPUSH fruits “apple”
```

Trong lệnh này, "apple" được thêm vào đầu danh sách được đại diện bởi khóa "fruits".

2. Lệnh RPOP loại bỏ và trả về giá trị cuối cùng của danh sách. Điều này hữu ích khi bạn sử dụng một danh sách như một ngăn xếp và muốn lấy ra phần tử được thêm vào gần đây nhất.

```
redis-cli RPOP fruits
```

Lệnh này sẽ loại bỏ và trả về giá trị "apple" từ danh sách "fruits".

3. **Hash**

Trong Redis Hash, một khóa chuỗi được ánh xạ vào một giá trị chuỗi. Tập hợp này của các cặp khóa-giá trị đặc biệt thích hợp để đại diện cho các đối tượng. Ví dụ, bạn có thể đại diện cho một đối tượng người dùng trong đó mỗi trường tương ứng với một thuộc tính của người dùng, như tên, email, hoặc tuổi.

Những trường này, hoặc khóa, là duy nhất trong mỗi hash. Tuy nhiên, cùng một trường có thể tồn tại trong nhiều hash riêng biệt. Hơn nữa, không thể sử dụng các cấu trúc dữ liệu phức tạp khác như Sets, Lists, hoặc Hashes bổ sung như giá trị.

Lệnh HSET được sử dụng để đặt một trường (hoặc khóa) thành một giá trị cụ thể trong một hash. Nếu trường đã tồn tại trong hash, giá trị của nó sẽ bị ghi đè.

```
redis-cli HSET user:1 name “Alice”
```

Ở đây, trong hash được đại diện bởi "user:1", chúng ta đang đặt trường "name" có giá trị là "Alice".

Lệnh HGET lấy giá trị được liên kết với một trường cụ thể trong một hash.

```
redis-cli HGET user:1 name
```

Sử dụng ví dụ trước đó, lệnh này lấy giá trị "Alice" được liên kết với trường "name" trong hash được đại diện bởi "user:1".

Lệnh HGETALL lấy tất cả các trường và giá trị trong một hash.

```
redis-cli HGETALL myhash
```

Điều này truy xuất tất cả các trường và giá trị từ hash được lưu trữ trong "myhash".

4. **Set**

Một Set trong Redis là một bộ sưu tập không thứ tự các chuỗi trong đó mỗi chuỗi là duy nhất. Điều này có nghĩa là không có mục nhập trùng lặp trong một tập hợp Redis. Tính duy nhất này của tập hợp khiến chúng hữu ích khi bạn cần theo dõi một bộ sưu tập các mục mà không có sự lặp lại, như việc theo dõi các thẻ người dùng, địa chỉ IP, hoặc thậm chí là các ID người dùng cho các khách truy cập duy nhất. Lệnh SADD được sử dụng để thêm một hoặc nhiều giá trị vào một tập hợp. Nếu một giá trị đã tồn tại trong tập hợp, nó sẽ không được thêm vào lại, giữ cho tính duy nhất của các giá trị trong tập hợp.

```
redis-cli SADD tags “music”
```

Trong lệnh này, giá trị "music" được thêm vào tập hợp được đại diện bởi khóa "tags". Nếu "music" đã tồn tại trong tập hợp này, tập hợp sẽ không thay đổi.

Lệnh SMEMBERS lấy tất cả các thành viên hoặc giá trị của một tập hợp.

```
redis-cli SMEMBERS tags
```

Sử dụng lệnh trước đó làm tham chiếu, lệnh này lấy tất cả các thành viên của tập hợp "tags". Trong trường hợp của chúng ta, nó sẽ trả về "music" giữa các thẻ có thể khác.

Lệnh SISMEMBER kiểm tra xem một giá trị có phải là thành viên của một tập hợp không.

```
redis-cli SISMEMBER myset "value"
```

Lệnh này kiểm tra xem "value" có tồn tại trong tập hợp "myset" không.

Lệnh SREM loại bỏ thành viên được chỉ định khỏi một tập hợp.

```
redis-cli SREM myset "value"
```

Lệnh này loại bỏ "value" khỏi tập hợp "myset".

5. **Sorted Set**

Sorted sets, hay còn được biết đến là Zsets (Z cho "Z-scored"), là một bộ sưu tập các thành viên duy nhất giống như các tập hợp. Tuy nhiên, điều làm cho chúng trở nên đặc biệt là sự liên kết của mỗi thành viên với một điểm số là một số thực.

Các thành viên trong một Sorted Set luôn được lấy theo thứ tự dựa trên điểm số của họ, điều này cung cấp một thứ tự nhất quán. Cấu trúc này đặc biệt hữu ích khi bạn cần duy trì một thứ tự các phần tử dựa trên một cơ chế xếp hạng hoặc điểm số như bảng xếp hạng trong các trò chơi, thứ hạng trang trong kết quả tìm kiếm, hoặc thậm chí là dữ liệu chuỗi thời gian.

1. Lệnh ZADD được sử dụng để thêm một thành viên với điểm số của nó vào sorted set. Nếu thành viên đã tồn tại, điểm số của nó sẽ được cập nhật.

```
redis-cli ZADD leaderboard 100 “player1”
```

Trong lệnh này, thành viên "player1" được thêm vào sorted set "leaderboard" với điểm số là 100. Nếu "player1" đã tồn tại, điểm số của nó sẽ được cập nhật thành 100.

2. Lệnh ZRANGE truy xuất các thành viên của sorted set trong một khoảng điểm số cụ thể.

```
redis-cli ZRANGE leaderboard 0 -1
```

Lệnh này lấy tất cả các thành viên của sorted set "leaderboard" từ đầu (0) đến cuối (-1) theo thứ tự tăng dần của điểm số của họ.

3. Lệnh ZSCORE trả về điểm số của một thành viên được chỉ định trong sorted set.

```
redis-cli ZSCORE leaderboard “player1”
```

Dựa trên các lệnh trước đó của chúng ta, điều này sẽ lấy điểm số của "player1" từ sorted set "leaderboard", là 100.

Lệnh ZRANGEBYSCORE trả về các thành viên trong một phạm vi điểm số cho trước.

```

redis-cli ZRANGEBYSCORE myzset 0 100

```

Lệnh này lấy các thành viên từ "myzset" có điểm số nằm trong khoảng từ 0 đến 100.

Lệnh ZREM loại bỏ các thành viên khỏi một sorted set.

```

redis-cli ZREM myzset "member"

```

Điều này loại bỏ "member" khỏi sorted set "myzset".

6. **HyperLoglog**
   HyperLogLog là một kiểu dữ liệu trong Redis được sử dụng để ước lượng số lượng phần tử duy nhất trong một tập hợp lớn mà không cần lưu trữ tất cả các phần tử đó. Nó cung cấp một cách hiệu quả và không đổi để ước lượng số lượng phần tử duy nhất trong một tập hợp lớn dựa trên một mẫu dữ liệu.

   HyperLogLog sử dụng một cấu trúc dữ liệu xấp xỉ để ước lượng sự đa dạng của các phần tử trong tập hợp, thường được biểu diễn bằng một chuỗi bit với độ dài cố định. Mỗi giá trị đầu vào được băm và sử dụng để thiết lập các bit trong chuỗi, từ đó tạo ra một ước lượng về số lượng phần tử duy nhất.

   Ưu điểm chính của HyperLogLog là nó tiết kiệm không gian lưu trữ so với việc lưu trữ tất cả các phần tử riêng lẻ trong tập hợp lớn. Nó cung cấp một ước lượng gần đúng về số lượng phần tử duy nhất với độ chính xác được kiểm soát, thường trong khoảng từ 1% đến 2%.

   Các lệnh cơ bản được sử dụng với HyperLogLog trong Redis bao gồm:

PFADD: Thêm một giá trị vào cấu trúc HyperLogLog.
PFCOUNT: Đếm số lượng phần tử duy nhất ước lượng trong cấu trúc HyperLogLog.
PFMERGE: Kết hợp nhiều cấu trúc HyperLogLog thành một cấu trúc duy nhất.

HyperLogLog thường được sử dụng trong các trường hợp mà việc lưu trữ số lượng phần tử duy nhất của một tập hợp lớn là quan trọng, như đếm số lượt truy cập trang web, đếm số lượng người dùng duy nhất hoặc đếm số lượt xem video.

V. Pub/Sub

Redis Publish/Subscribe (Pub/Sub) là một mô hình truyền thông trong đó các client Redis có thể gửi (publish) các thông điệp và đăng ký (subscribe) để nhận các thông điệp đó. Điều này cho phép các ứng dụng xây dựng một hệ thống giao tiếp linh hoạt và phân tán, nơi mà các thành phần có thể giao tiếp với nhau mà không cần biết về sự tồn tại của nhau.

Cơ bản, có hai loại client trong Pub/Sub:

Publisher: Client gửi các thông điệp đến một hoặc nhiều channels (kênh).
Subscriber: Client đăng ký nhận thông điệp từ một hoặc nhiều channels.

Một số điểm chính của Redis Pub/Sub bao gồm:

Channels (Kênh): Các thông điệp được phát đi và nhận được qua các channels. Mỗi channel là một chuỗi định danh mà các client có thể đăng ký để nhận thông điệp.

Patterns (Mẫu): Ngoài việc đăng ký theo từng channel cụ thể, client cũng có thể đăng ký theo một hoặc nhiều mẫu (patterns). Khi một thông điệp được gửi đến một channel khớp với mẫu, tất cả các client đã đăng ký theo mẫu đó sẽ nhận được thông điệp.

Công bố (Publish): Publisher gửi thông điệp tới một channel cụ thể.

Đăng ký (Subscribe): Subscriber đăng ký để nhận các thông điệp từ một hoặc nhiều channels hoặc theo các mẫu.

Hủy đăng ký (Unsubscribe): Subscriber có thể hủy đăng ký để không nhận thông điệp từ một hoặc nhiều channels hoặc theo các mẫu nữa.

Công nghệ Publish/Subscribe của Redis thích hợp cho nhiều trường hợp sử dụng như thông báo thời gian thực, phân phối dữ liệu, xử lý sự kiện, và nhiều ứng dụng phân tán khác. Đặc biệt, nó có thể được tích hợp vào các hệ thống phân tán mà các thành phần cần liên lạc với nhau mà không phụ thuộc vào cấu trúc hoặc vị trí của họ.

VI. Stream

Redis Streams là một kiểu dữ liệu mới được giới thiệu từ phiên bản Redis 5.0 trở lên, giúp hỗ trợ việc xử lý và lưu trữ dữ liệu dạng log theo thời gian thực. Streams cho phép bạn lưu trữ các thông điệp có thứ tự theo thời gian và truy xuất chúng dựa trên các thuộc tính như ID, group, hoặc range của thời gian.

Một số khái niệm quan trọng trong Redis Streams:

1. **Stream**: Một stream là một chuỗi các mục dữ liệu được đặt theo thứ tự thời gian. Mỗi mục được gọi là một entry.

2. **Entry**: Mỗi entry trong stream chứa hai phần chính: một ID duy nhất và một cấu trúc dữ liệu, thường là một từ điển (dictionary), chứa các trường và giá trị của dữ liệu.

3. **ID**: Mỗi entry trong stream có một ID duy nhất, đảm bảo tính duy nhất và thứ tự của mỗi entry.

4. **Consumer group**: Một consumer group là một nhóm các consumer (người tiêu thụ) của các dữ liệu trong một stream. Consumer groups cho phép một nhóm consumer chia sẻ việc tiêu thụ các entry từ stream một cách song song.

Các lệnh cơ bản được sử dụng với Redis Streams:

1. **XADD**: Thêm một entry mới vào stream.

2. **XREAD**: Đọc các entry từ một hoặc nhiều stream.

3. **XGROUP**: Quản lý consumer groups.

4. **XREADGROUP**: Đọc các entry từ một stream với sự hỗ trợ của một consumer group.

5. **XINFO**: Cung cấp thông tin về stream và consumer groups.

Redis Streams thích hợp cho các ứng dụng như xử lý dữ liệu thời gian thực, gửi thông báo, xử lý sự kiện, và quản lý log. Đặc biệt, nó cung cấp tính năng phân tán cao và có thể được tích hợp vào các hệ thống phân tán một cách linh hoạt.