What on Earth is Redis?

MLH, November 2023

Justin Castilla
justin@redis.com
https://university.redis.com

What's going on?

We'll cover:

- Recent Pokemon GO use case
- Redis Design & Data Type basics
- Example Use Cases with Redis
    - Implementing a queue with Lists
    - Player leaderboard with Sorted Sets
    - Caching data with Strings and Key Expiry
    - Querying and Indexing with JSON and Search

github.com/redislabs-training/mlh-redis-intro

Pokemon GO + Redis = <3

Challenge: As thousands of Pokémon GO players participate in Raid Battles, Niantic's servers had become bogged down during the preparation phase when people form and join teams. Niantic needed a fast, responsive database that scales quickly to accommodate surges in Pokémon GO activity.

Solution: To support heightened player activity, Niantic caches high volumes of game data in a Redis cluster. All Pokémon GO servers can access this shared data, reducing latency and boosting performance for multi-player Raid events.

Pokemon GO + Redis = <3

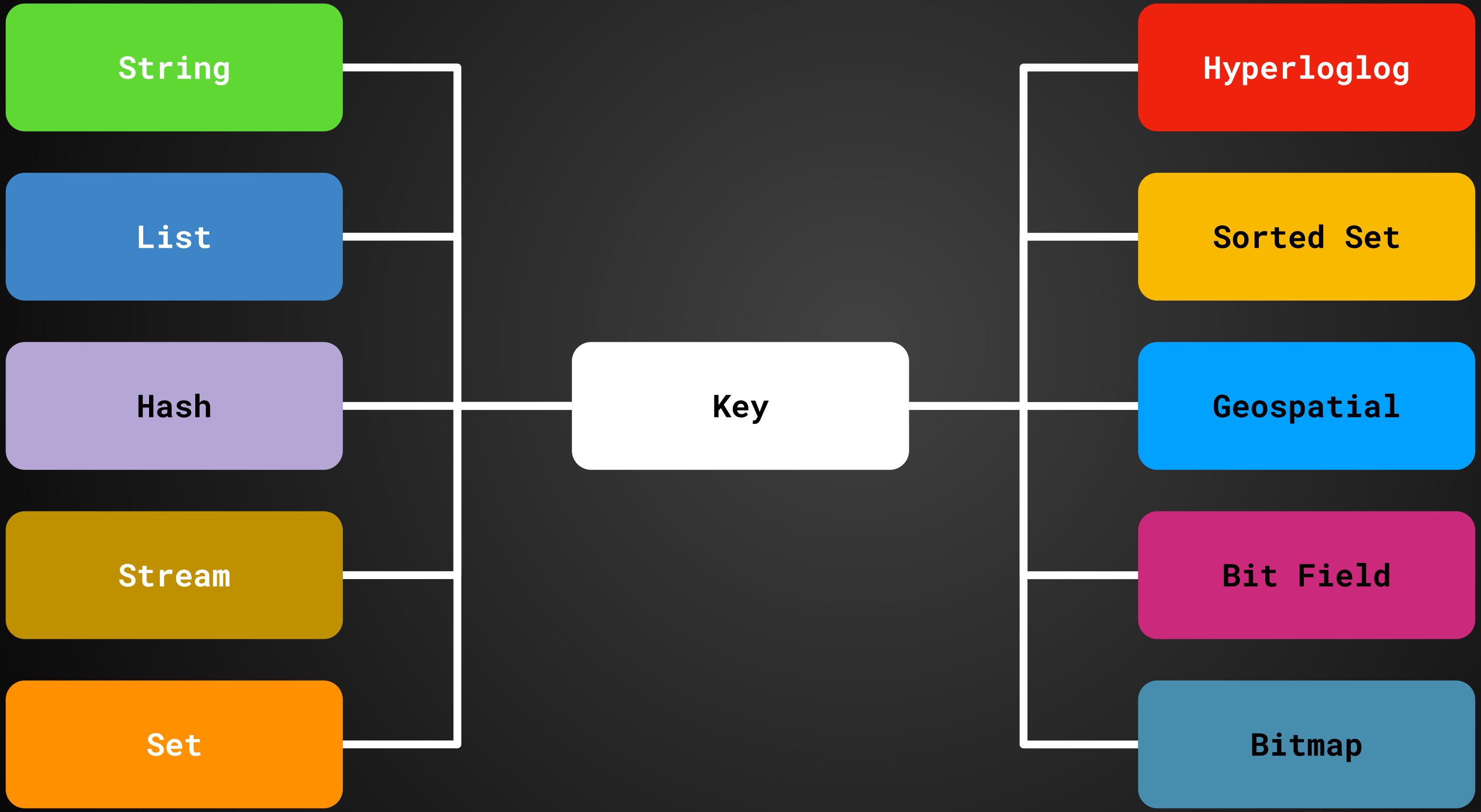
What does this mean?

Redis servers are created whenever a Raid Battle starts. User data is migrated to the new server, along with specific team data, user trends, and relationship data.

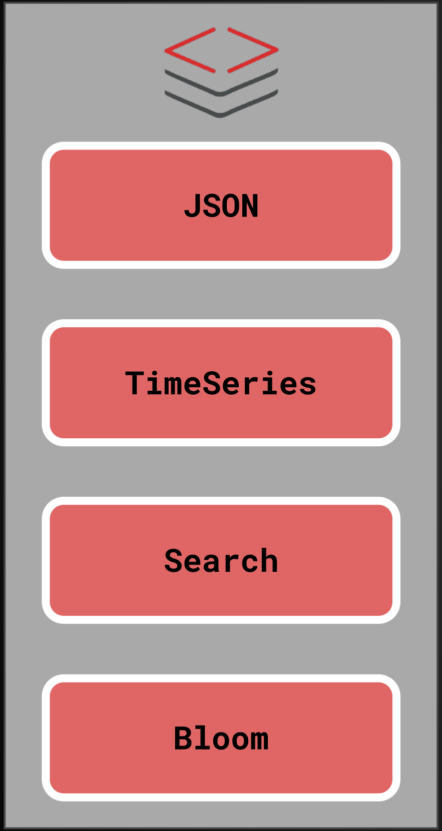
To keep the interaction fast, the servers are created as close to the actual Redis Battle as possible.

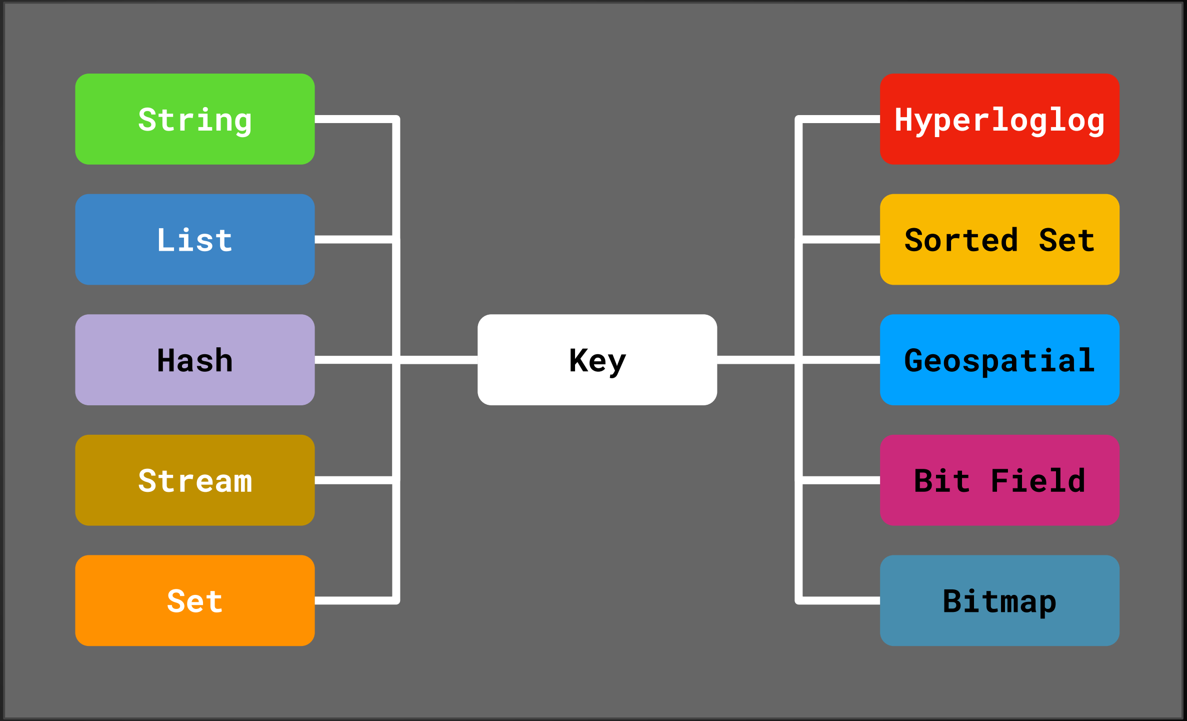
Data is stored as JSON, with actions stored in queues. Image data, metadata, HP, CP, user levels, and more are stored temporarily in these extra servers for the fastest retrieval. Using normal remote storage or disks would slow the experience down considerably.

OSS Data Structures / Data Types

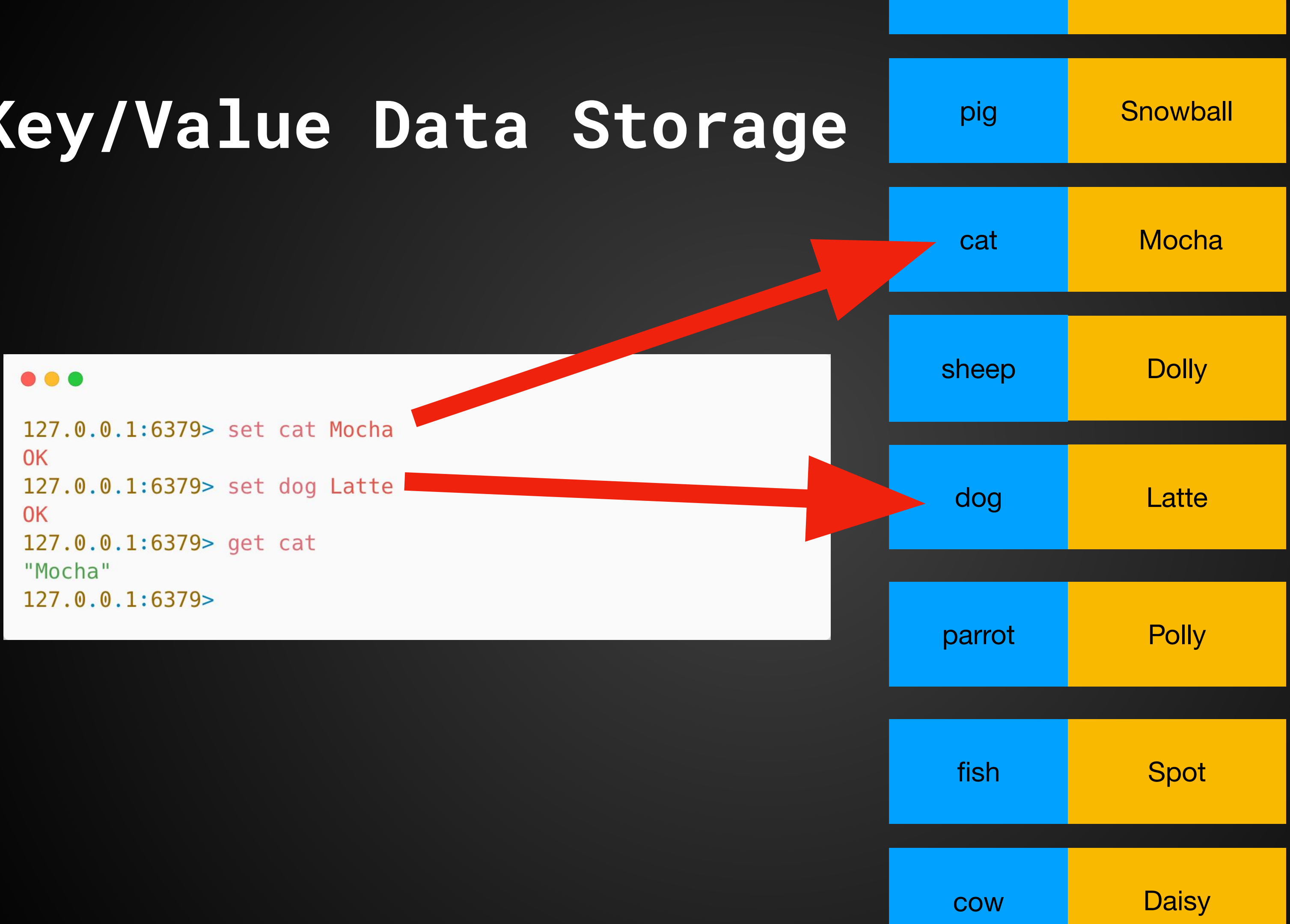


Redis Stack Data Structures Extension





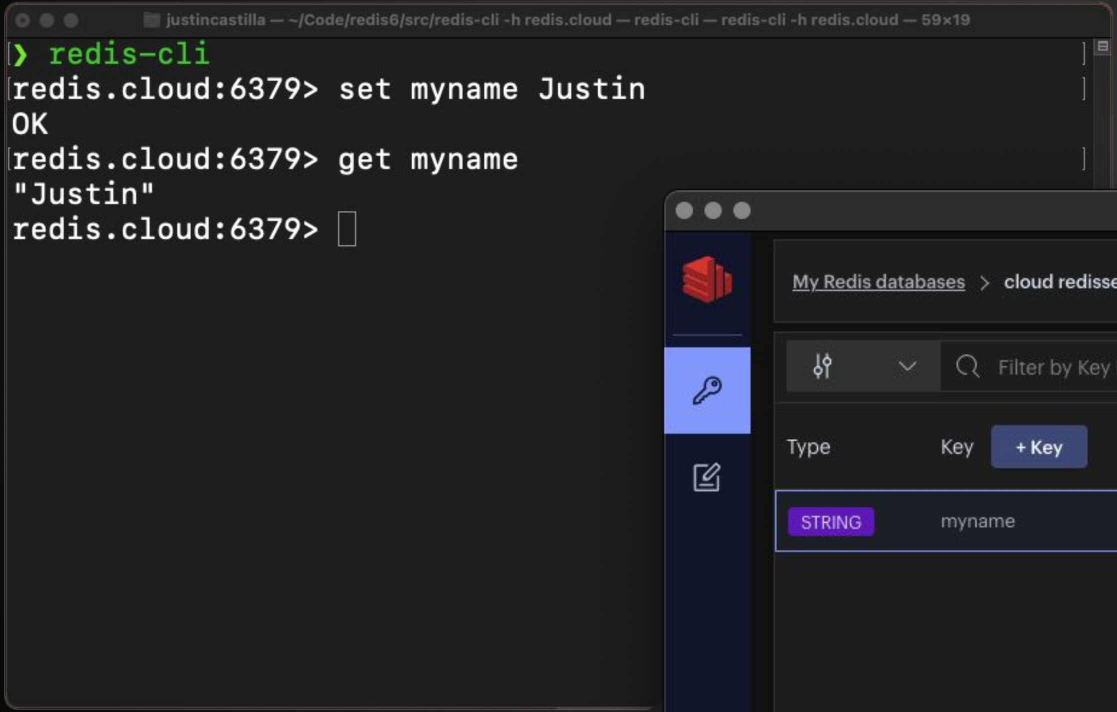
Key/Value Data Storage



```
127.0.0.1:6379> set cat Mocha
OK
127.0.0.1:6379> set dog Latte
OK
127.0.0.1:6379> get cat
"Mocha"
127.0.0.1:6379>
```

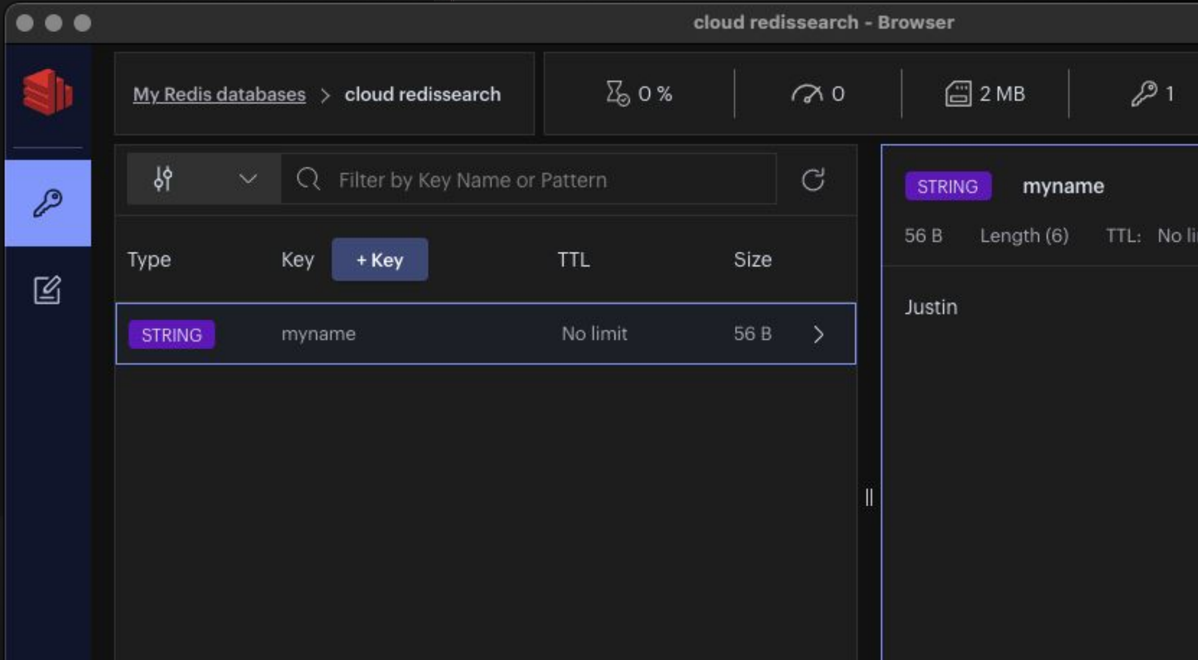
pig	Snowball
cat	Mocha
sheep	Dolly
dog	Latte
parrot	Polly
fish	Spot
cow	Daisy

Sending Commands to Redis



justincastilla — ~/Code/redis6/src/redis-cli -h redis.cloud — redis-cli — redis-cli -h redis.cloud — 59×19

```
❯ redis-cli
redis.cloud:6379> set myname Justin
OK
redis.cloud:6379> get myname
"Justin"
redis.cloud:6379> 
```



cloud redissearch - Browser

My Redis databases > cloud redissearch

0 % | 0 | 2 MB | 1

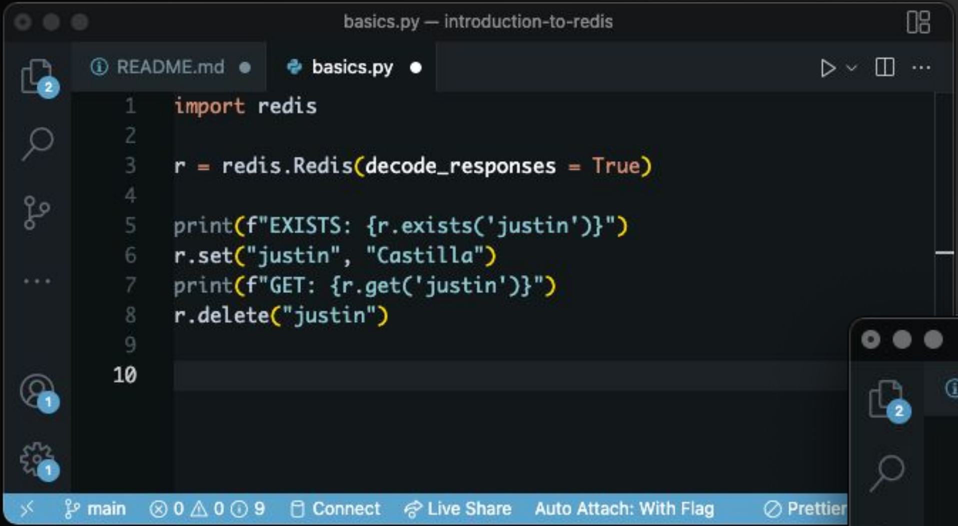
Type	Key	TTL	Size
STRING	myname	No limit	56 B

STRING myname

56 B Length (6) TTL: No li

Justin

Using a Redis Client Library

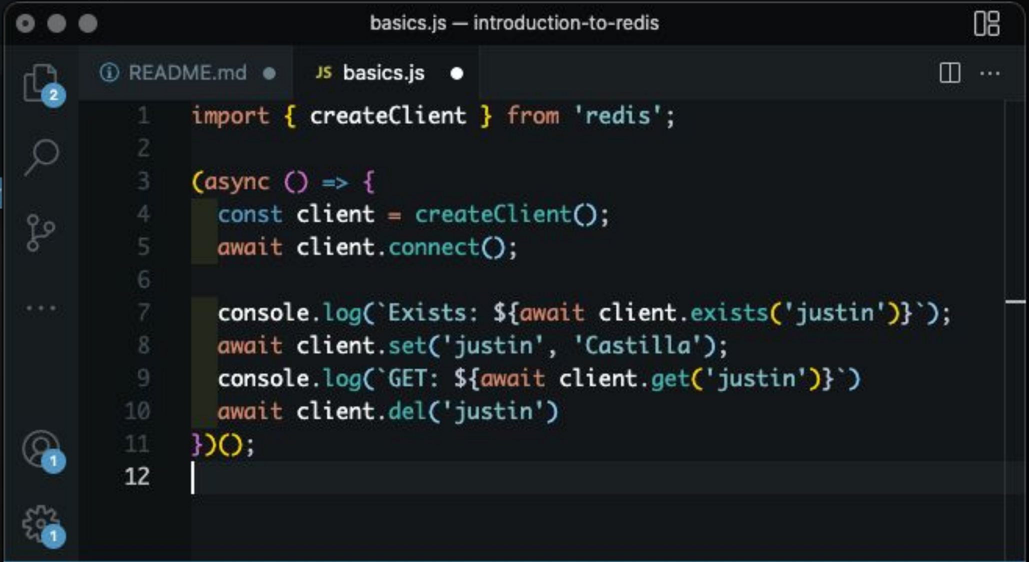


basics.py — introduction-to-redis

```python
import redis

r = redis.Redis(decode_responses = True)

print(f"EXISTS: {r.exists('justin')}")
r.set("justin", "Castilla")
print(f"GET: {r.get('justin')}")
r.delete("justin")
```

redis-py

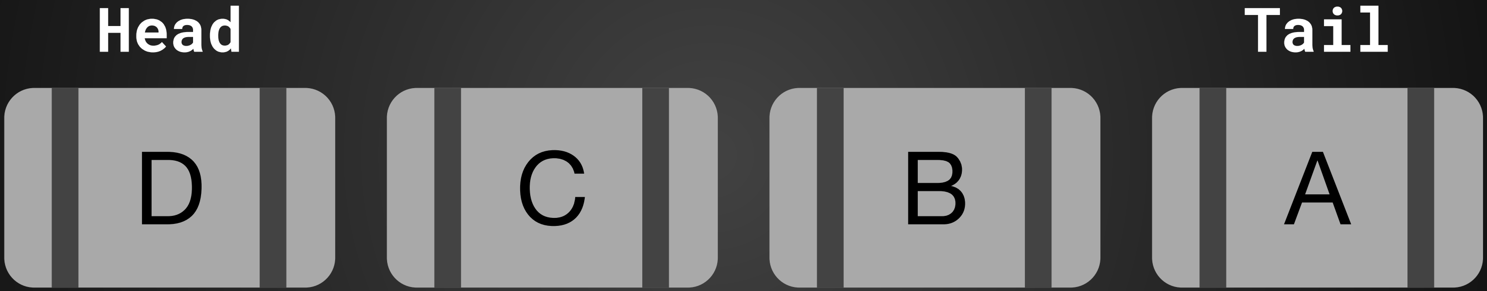
node-redis



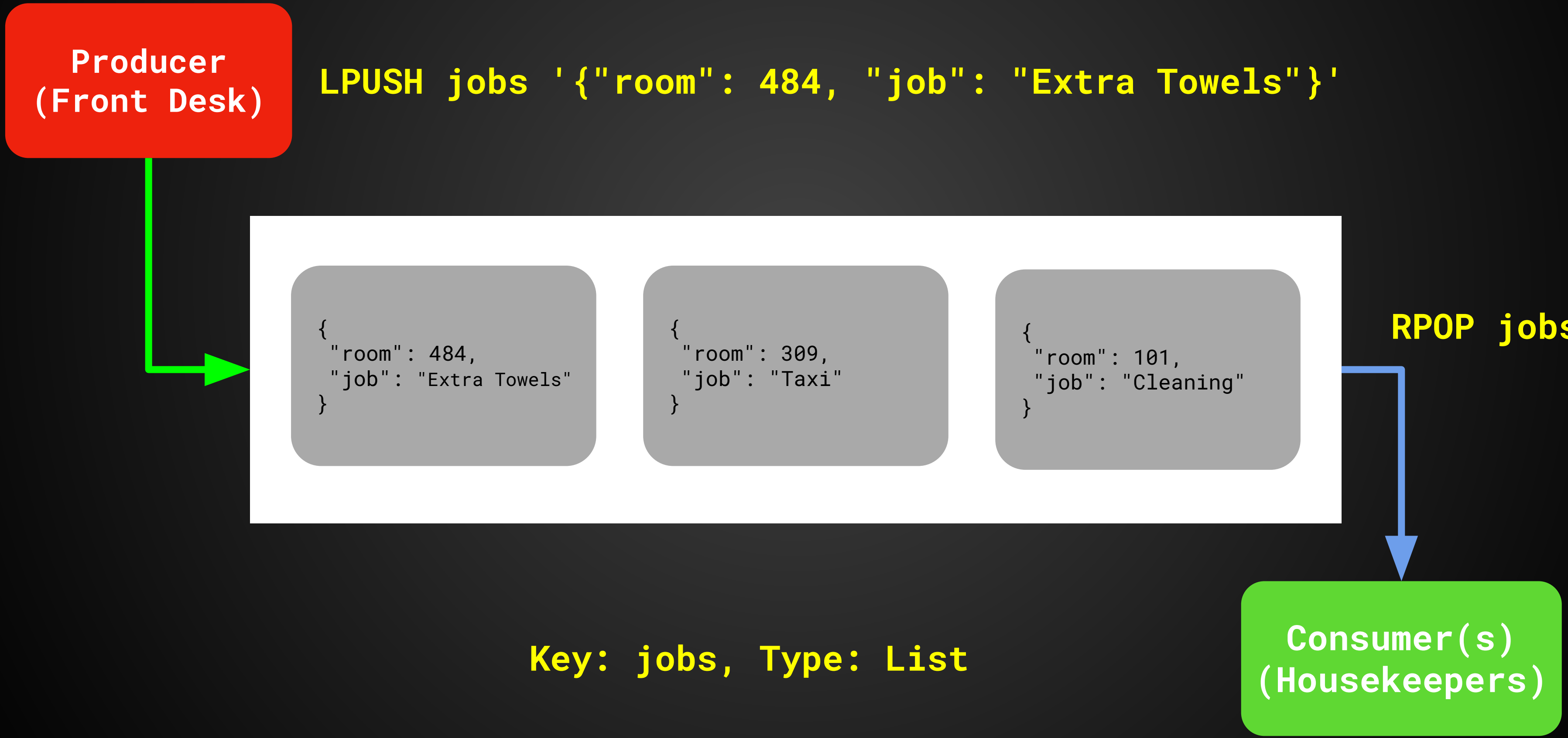
basics.js — introduction-to-redis

```javascript
import { createClient } from 'redis';

(async () => {
  const client = createClient();
  await client.connect();

  console.log(`Exists: ${await client.exists('justin')}`);
  await client.set('justin', 'Castilla');
  console.log(`GET: ${await client.get('justin')}`)
  await client.del('justin')
})();
```

Use Case: Queuing with Lists



Queuing with Lists: Housekeeping



```
LPUSH jobs '{"room": 484, "job": "Extra Towels"}'
```

```
{
  "room": 484,
  "job": "Extra Towels"
}
```

```
{
  "room": 309,
  "job": "Taxi"
}
```

```
{
  "room": 101,
  "job": "Cleaning"
}
```

Key: jobs, Type: List

Use Case: Leaderboard with Sorted Sets

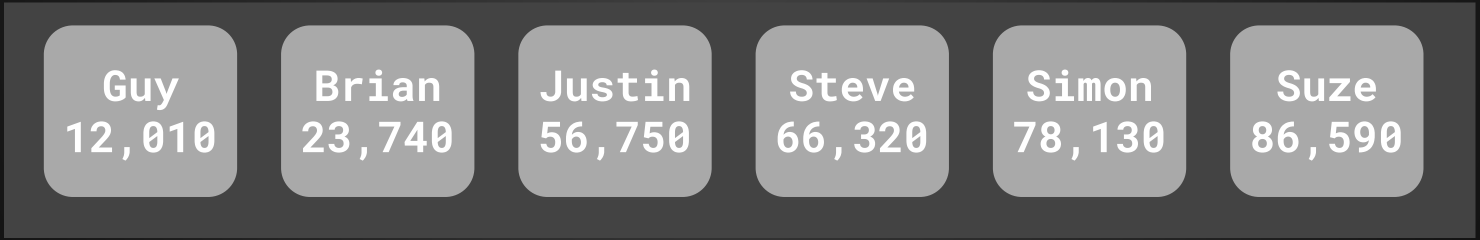
What is a sorted set?

- A collection of unique values
- Not a list, where duplicates can exist
- If you add an element that already exists, it will be ignored
- A score is associated with each element (by you)
- Automagically sorted upon insertion from lowest to highest

Use Case: Leaderboard with Sorted Sets

Let's make a sorted set of scores

- ZADD - adds an element and its score to an existing or new sorted set
- ZRANGE - returns a portion of the set based on a range and other options

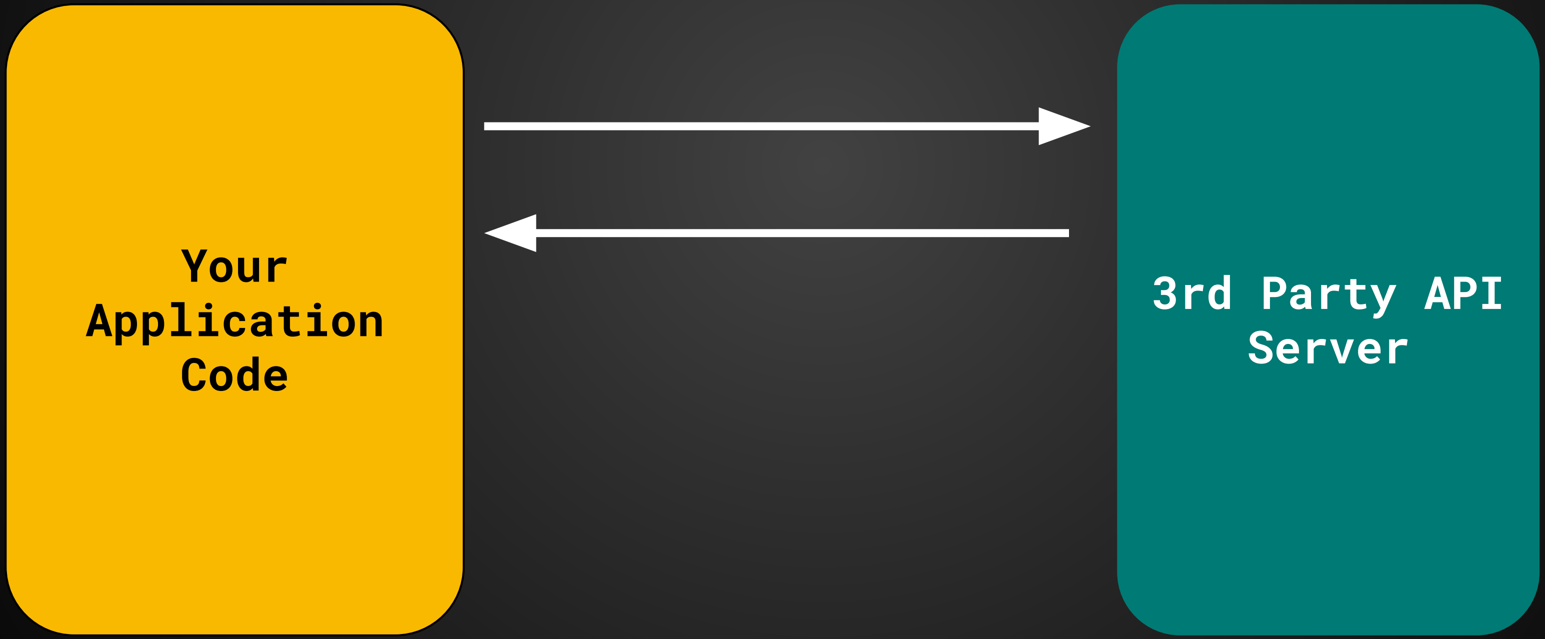


Use Case: Caching

Wikipedia: *“A cache is a hardware or software component that stores data so that future requests for that data can be served faster; the data stored in a cache might be the result of an earlier computation or a copy of data stored elsewhere.”*

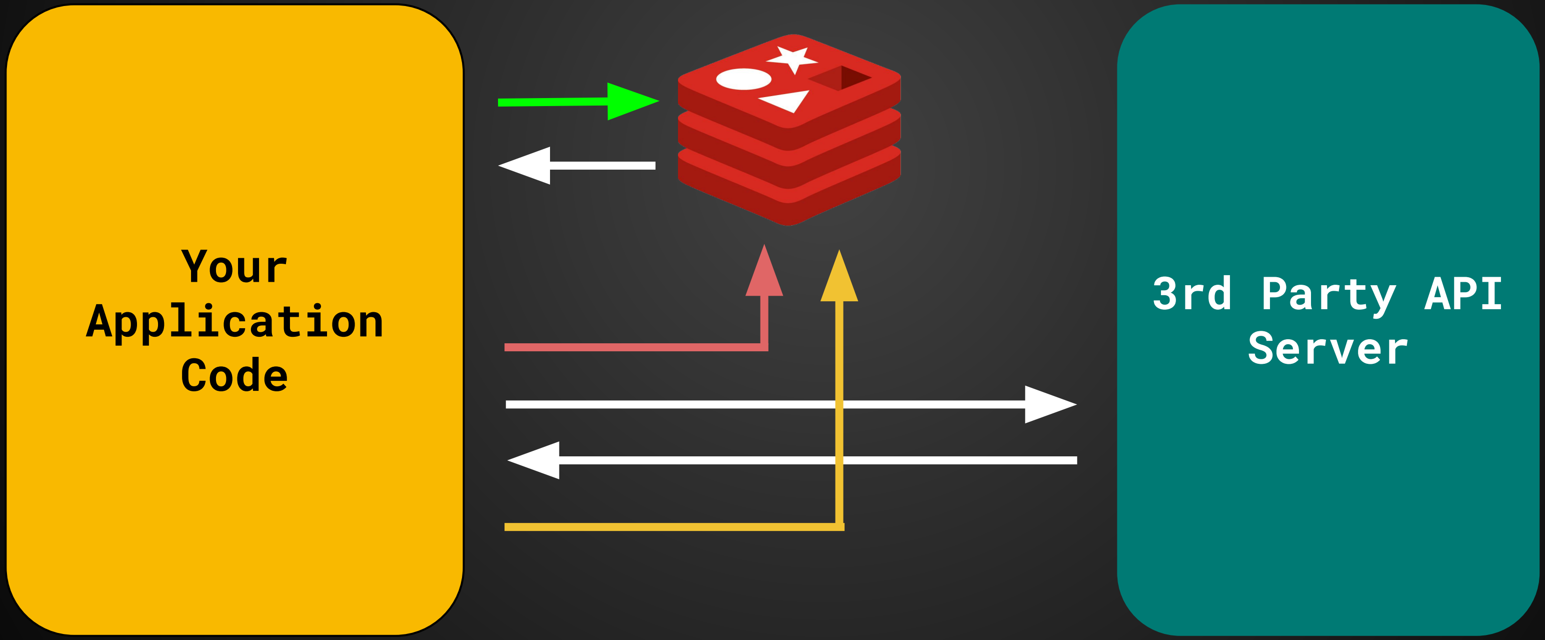
Caching a 3rd Party API Response

Without a cache…



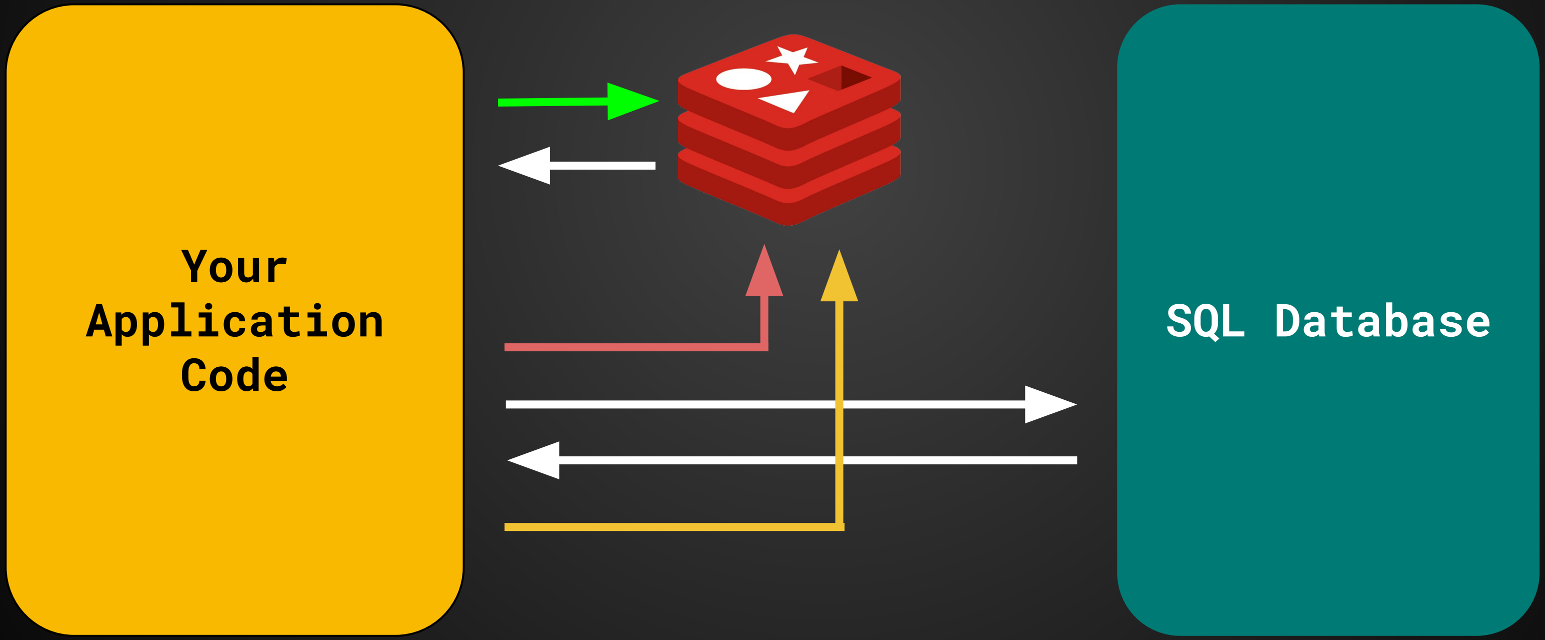
Caching a 3rd Party API Response

Using Redis as a cache…



Caching a SQL Query

Using Redis as a cache…



Caching Commands

Caching Logic:

1. Check Redis for the value using
   `GET <key>`
2. If it exists, great!
3. If it doesn't exist, then fetch the data from the original source.
4. Return the data, then create an entry in Redis with the following command:
   `SETEX <key> <seconds> <value>`
5. Setting a TTL ensures your data will be fresh and updated!

Caching examples

Node.js Caching API Responses Example:
bit.ly/49nXewg

Node.js Caching Slow SQL Queries Example:
bit.ly/40ulGao

JSON capabilities

- Redis Stack stores and retrieves JSON as a native data type!
- Stores Lists, Strings, Numbers, Booleans, and Objects like normal JSON
- Reduces the need to convert your JSON to and from strings or SQL rows
- Retrieve all or portions of your JSON code
- Uses common commands (GET, SET, etc.)
- Still very VERY fast!

Search capabilities

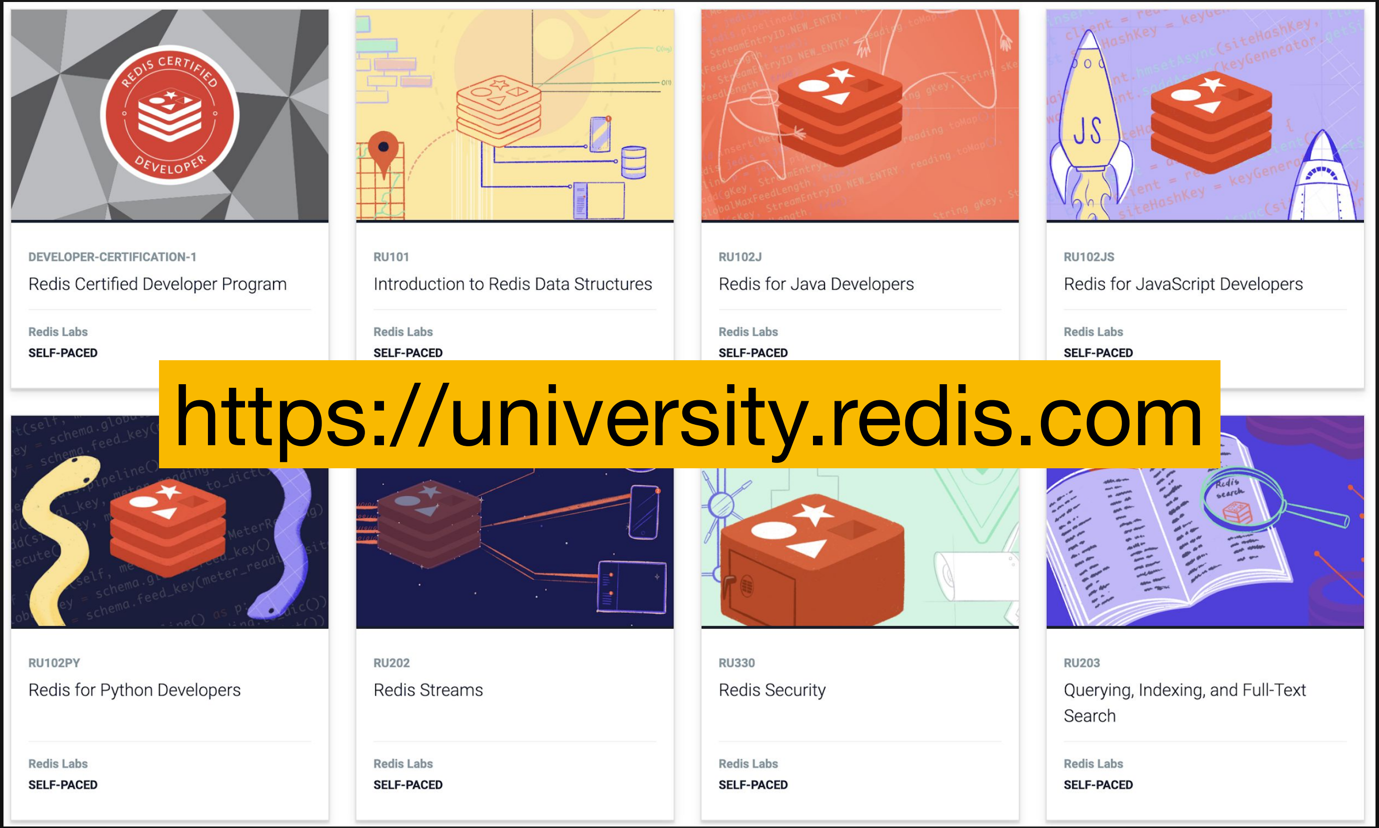
- Redis Stack facilitates search of Hashes and JSON in Redis
- Search full text, strings, tags, geospatial coordinates, and numbers
- You define what fields to search, and what keys to index
- All previous and future keys with your defined pattern are automagically indexed
- Great for autocomplete or a VERY fast search feature.

Search with JSON example

Node.js Music Discography Example:
bit.ly/3BLSeBo

Python Address Book Example:
bit.ly/49jp90w

Would You Like to Know More?



DEVELOPER-CERTIFICATION-1
Redis Certified Developer Program
Redis Labs
SELF-PACED

RU101
Introduction to Redis Data Structures
Redis Labs
SELF-PACED

RU102J
Redis for Java Developers
Redis Labs
SELF-PACED

RU102JS
Redis for JavaScript Developers
Redis Labs
SELF-PACED

RU102PY
Redis for Python Developers
Redis Labs
SELF-PACED

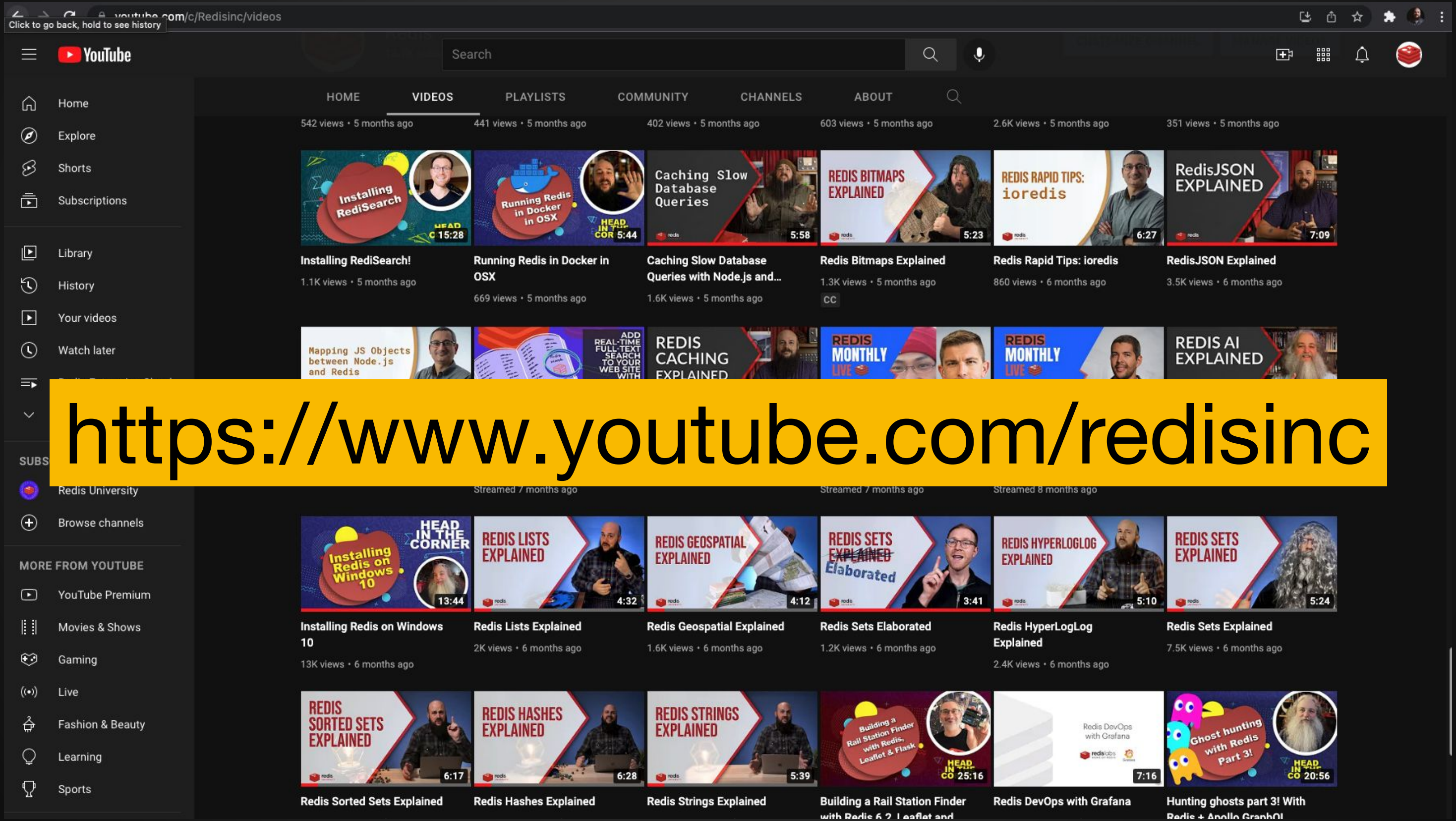
RU202
Redis Streams
Redis Labs
SELF-PACED

RU330
Redis Security
Redis Labs
SELF-PACED

RU203
Querying, Indexing, and Full-Text Search
Redis Labs
SELF-PACED

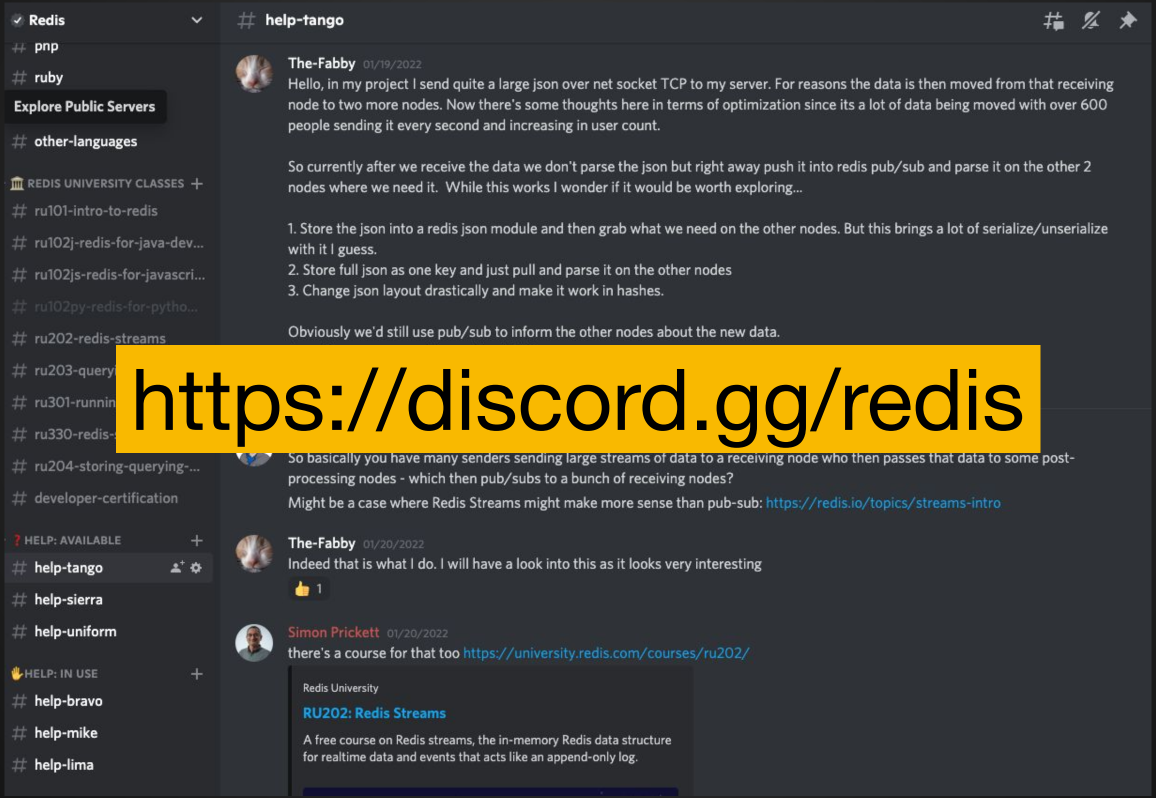
https://university.redis.com

Would You Like to Know More?


youtube.com/c/Redisinc/videos
Click to go back, hold to see history
YouTube
Search
HOME
VIDEOS
PLAYLISTS
COMMUNITY
CHANNELS
ABOUT
Home
Explore
Shorts
Subscriptions
Library
History
Your videos
Watch later
SUBS
Redis University
Browse channels
MORE FROM YOUTUBE
YouTube Premium
Movies & Shows
Gaming
Live
Fashion & Beauty
Learning
Sports
Installing RediSearch!
Running Redis in Docker in OSX
Caching Slow Database Queries with Node.js and...
Redis Bitmaps Explained
Redis Rapid Tips: ioredis
RedisJSON Explained
Installing Redis on Windows 10
Redis Lists Explained
Redis Geospatial Explained
Redis Sets Elaborated
Redis HyperLogLog Explained
Redis Sets Explained
Redis Sorted Sets Explained
Redis Hashes Explained
Redis Strings Explained
Building a Rail Station Finder with Redis 6.2, Leaflet and
Redis DevOps with Grafana
Hunting ghosts part 3! With Redis + Apollo GraphQL

https://www.youtube.com/redisinc

Would You Like to Know More?



The-Fabby 01/19/2022
Hello, in my project I send quite a large json over net socket TCP to my server. For reasons the data is then moved from that receiving node to two more nodes. Now there's some thoughts here in terms of optimization since its a lot of data being moved with over 600 people sending it every second and increasing in user count.

So currently after we receive the data we don't parse the json but right away push it into redis pub/sub and parse it on the other 2 nodes where we need it. While this works I wonder if it would be worth exploring...

1. Store the json into a redis json module and then grab what we need on the other nodes. But this brings a lot of serialize/unserialize with it I guess.
2. Store full json as one key and just pull and parse it on the other nodes
3. Change json layout drastically and make it work in hashes.

Obviously we'd still use pub/sub to inform the other nodes about the new data.

So basically you have many senders sending large streams of data to a receiving node who then passes that data to some post-processing nodes - which then pub/subs to a bunch of receiving nodes?
Might be a case where Redis Streams might make more sense than pub-sub: https://redis.io/topics/streams-intro

The-Fabby 01/20/2022
Indeed that is what I do. I will have a look into this as it looks very interesting
👍 1

Simon Prickett 01/20/2022
there's a course for that too https://university.redis.com/courses/ru202/

Redis University
RU202: Redis Streams
A free course on Redis streams, the in-memory Redis data structure for realtime data and events that acts like an append-only log.

https://discord.gg/redis

Thank You!

http://github.com/justincastilla/introduction-to-redis

Justin Castilla
@justcastilla
justin@redis.com
https://university.redis.com
https://developer.redis.com
https://discord.gg/redis