**Difference between In-memory dbs and In-memory cache**

**Cache** –

By definition means it is stored in memory. Any data stored in memory (RAM) for faster access is called cache.

Examples: Ehcache, Memcache Typically you put an object in cache with String as Key and access the cache using the Key. It is very straight forward. It depends on the application when to access the cache vs database and no complex processing happens in the Cache. If the cache spans multiple machines, then it is called distributed cache. For example, **Netflix** uses EVCAche which is built on top of Memcache to store the users movie recommendations that you see on the home screen.

**In Memory Database** –

It has all the features of a Cache plus come processing/querying capabilities. Redis falls under this category. Redis supports multiple data structures and you can query the data in the Redis ( examples like get last 10 accessed items, get the most used item etc). It can span multiple machine and is usually very high performant and also support persistence to disk if needed. For example, **Twitter** uses Redis database to store the timeline information.

**Redis**

Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker. Redis provides data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes, and streams.

Redis supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes with radius queries and streams. Redis has built-in replication, Lua scripting, LRU eviction, transactions and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster.

**Redis persistence**

1. RDB (Redis Database): The RDB persistence performs point-in-time snapshots of your dataset at specified intervals.
2. AOF (Append Only File): The AOF persistence logs every write operation received by the server, that will be played again at server start-up, reconstructing the original dataset. Commands are logged using the same format as the Redis protocol itself, in an append-only fashion. Redis is able to rewrite the log in the background when it gets too big.
3. No persistence: If you wish, you can disable persistence completely, if you want your data to just exist as long as the server is running.
4. RDB + AOF: It is possible to combine both AOF and RDB in the same instance. Notice that, in this case, when Redis restarts the AOF file will be used to reconstruct the original dataset since it is guaranteed to be the most complete.

**Redis usages**

**Session store**

Redis as an in-memory data store with high availability and persistence is a popular choice among application developers to store and manage [session data](https://aws.amazon.com/blogs/developer/elasticache-as-an-asp-net-session-store/) for internet-scale applications. Redis provides the sub-millisecond latency, scale, and resiliency required to manage session data such as user profiles, credentials, session state, and user-specific personalization.

**Rich media streaming**

Redis offers a fast, in-memory data store to power live streaming use cases. Redis can be used to store metadata about users' profiles and viewing histories, authentication information/tokens for millions of users, and manifest files to enable CDNs to stream videos to millions of mobile and desktop users at a time.

**Geospatial**

Redis offers purpose-built in-memory data structures and operators to manage real-time [geospatial data](https://aws.amazon.com/blogs/database/amazon-elasticache-utilizing-redis-geospatial-capabilities/) at scale and speed. Commands such as GEOADD, GEODIST, GEORADIUS, and GEORADIUSBYMEMBER to store, process, and analyze geospatial data in real-time make geospatial easy and fast with Redis. You can use Redis to add location-based features such as drive time, drive distance, and points of interest to your applications.

**Machine Learning**

Modern data-driven applications require machine learning to quickly process a massive volume, variety, and velocity of data and automate decision making. For use cases like fraud detection in gaming and financial services, real-time bidding in ad-tech, and matchmaking in dating and ride sharing, the ability to process live data and make decisions within tens of milliseconds is of utmost importance. Redis gives you a fast in-memory data store to build, train, and deploy machine learning models quickly.

**Real-time analytics**

Redis can be used with streaming solutions such as Apache Kafka and Amazon Kinesis as an in-memory data store to ingest, process, and analyze real-time data with sub-millisecond latency. Redis is an ideal choice for [real-time analytics](https://aws.amazon.com/elasticache/redis/#Real-time_Analytics) use cases such as social media analytics, ad targeting, personalization, and IoT.

## **Choosing between Redis and Memcached**

Redis and Memcached are popular, open-source, in-memory data stores. Although they are both easy to use and offer high performance, there are important differences to consider when choosing an engine. Memcached is designed for simplicity while Redis offers a rich set of features that make it effective for a wide range of use cases. Understand your requirements and what each engine offers to decide which solution better meets your needs.



|  |  |  |
| --- | --- | --- |
|  |  |  |

**Sub-millisecond latency**

Both Redis and Memcached support sub-millisecond response times. By storing data in-memory they can read data more quickly than disk based databases.

**Developer ease of use**

Both Redis and Memcached are syntactically easy to use and require a minimal amount of code to integrate into your application.

**Data partitioning**

Both Redis and Memcached allow you to distribute your data among multiple nodes. This allows you to scale out to better handle more data when demand grows.

Support for a broad set of programming languages

Both Redis and Memcached have many open-source clients available for developers. Supported languages include Java, Python, PHP, C, C++, C#, JavaScript, Node.js, Ruby, Go and many others.

**Advanced data structures**

In addition to strings, Redis supports lists, sets, sorted sets, hashes, bit arrays, and hyperloglogs. Applications can use these more advanced data structures to support a variety of use cases. For example, you can use Redis Sorted Sets to easily implement a game leaderboard that keeps a list of players sorted by their rank.

**Multithreaded architecture**

Since Memcached is multithreaded, it can make use of multiple processing cores. This means that you can handle more operations by scaling up compute capacity.

**Snapshots**

With Redis you can keep your data on disk with a point in time snapshot which can be used for archiving or recovery.

**Replication**

Redis lets you create multiple replicas of a Redis primary. This allows you to scale database reads and to have highly available clusters.

**Transactions**

Redis supports transactions which let you execute a group of commands as an isolated and atomic operation.

**Pub/Sub**

Redis supports Pub/Sub messaging with pattern matching which you can use for high performance [chat rooms](https://aws.amazon.com/blogs/database/how-to-build-a-chat-application-with-amazon-elasticache-for-redis/), real-time comment streams, social media feeds, and server intercommunication.

**Lua scripting**

Redis allows you to execute transactional Lua scripts. Scripts can help you boost performance and simplify your application.

**Geospatial support**

Redis has purpose-built commands for working with real-time [geospatial data](https://aws.amazon.com/blogs/database/amazon-elasticache-utilizing-redis-geospatial-capabilities/) at scale. You can perform operations like finding the distance between two elements (for example people or places) and finding all elements within a given distance of a point.

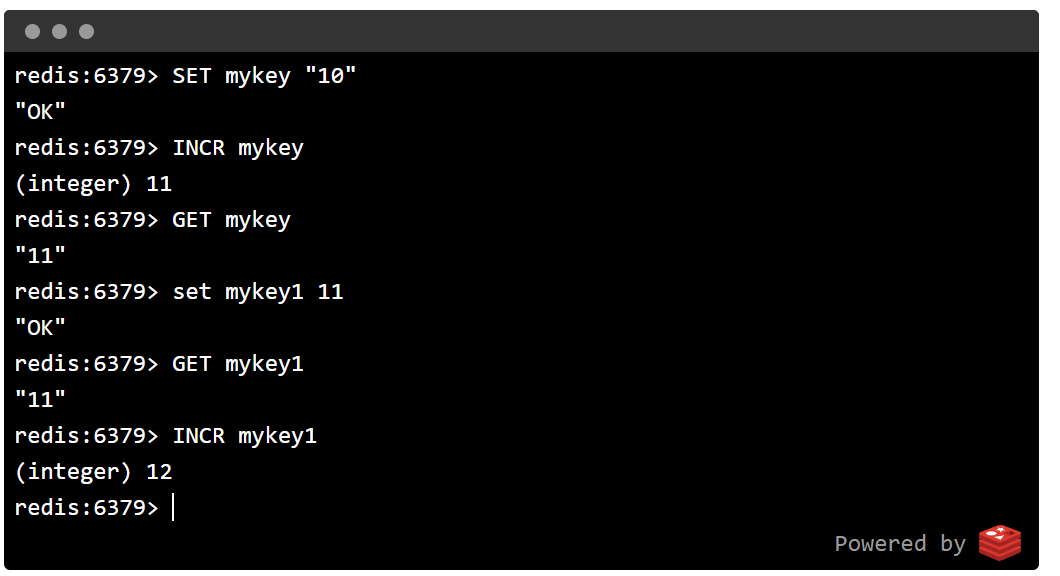
**String Data Type**

Redis string can contain any kind of data, for instance a JPEG image or a serialized Ruby object.

A String value can be at max 512 Megabytes in length.

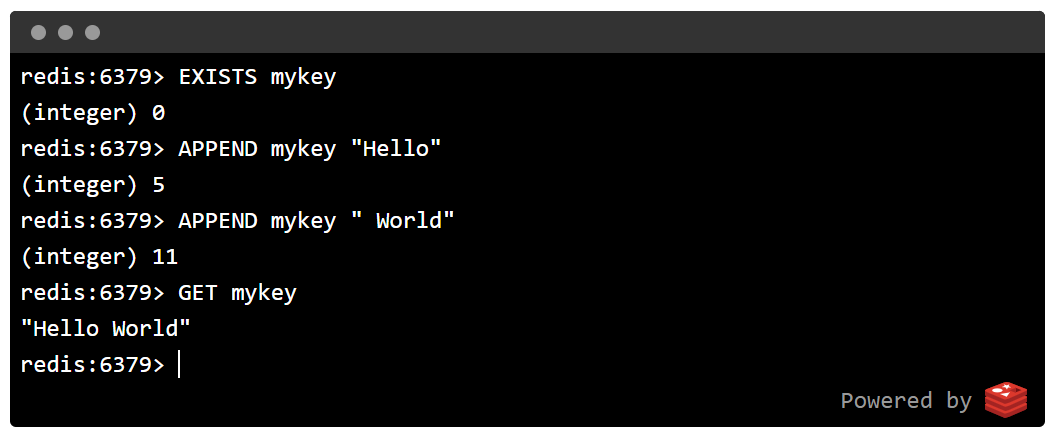
String representation integer can use [INCR](https://redis.io/commands/incr), [DECR](https://redis.io/commands/decr), [INCRBY](https://redis.io/commands/incrby) commands.

Append to strings with the [APPEND](https://redis.io/commands/append) command.



Text

Description automatically generated



A screenshot of a computer

Description automatically generated with medium confidence