# Redis

## What is Redis:

* Remote Dictionary Server is an in-memory database. (RAM or SSD)
* Mostly used as cache on top of other databases to improve performance
* Fully fledged primary database
* Store and persist multiple data formats for complex applications.

Diagram

Description automatically generated

## Use-cases and Benefits of a Multi-Model DB?

**Scenario**:

We have a complex application with multiple users and multiple microservices:

* it use MySQL to store the data.
* elasticsearch database for fast filtering
* neo4j Graph Database as users are connected together( represent connections).
* MongoDB Document Database for sharing media content.
* Redis cache service for better performance.

**Challenges faced**:

* Data services need to be deployed and maintained by team.
* Know how needed for each data service.
* Different scaling & infrastructure Requirements.

**Solution**:

1. Use managed data services by cloud providers:

* Expensive.
* Complex code on BE as multiple microservices talk to eachother(separate connectors between each database).
* Testing applications are challenging.
* Higher the microservice count the higher the latency due to connectors(network hopping).

1. Multi model Database like Redis:

* Run everything in one DB.
* Simpler.
* Reduced latency due to single DB(no network hops).
* Allows you to save different Data structure.
* No separate cache needed.

## How does it work?

Redis core which is a key value store storing all types of data like:

* Strings
* Sets
* Bitmaps
* Sorted Sets
* Bit Field
* Geospatial
* Hashes
* Hyperloglog
* Lists
* Streams

This can be extended by Modules for different data types according to application needs like:

Logo, company name

Description automatically generated

All of this is Modular which can be handpicked from redis website according to functionality. Redis has a Database Cache(out of the box Cache) so less complexity, no need to implement a caching layer for managing, populating and validating cache.

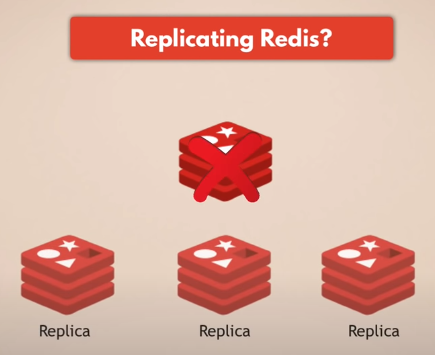
Being in-memory DB:

* Faster performance.
* Schemaless so Faster Tests (no need for initailizing schema).

Data Persistence & Durability?

As RAM is volatile memory there might be an issue for memory loss in that case:

* Redis database can be **Replicated to have memory backup.**



How an we keep the data safely persisted?

1. Snapshotting:

* Produces single-file point-in-time snapshots of your dataset
  + → stored in disks with can be recovered if data is gone.
  + → great for backups and disaster recovery but you may loose the lastest minutes of data.
* Configure based on time or the number of writes passed or number of requests etc.

1. Append only File(AOF)

* logs every write opreation continously.
* When redis is restarted redis will re-play the AOF to rebuild the state.
* Durable but slower than RDB.

1. Combination of both (snapshotting + AOF)

* Regular snapshots for DB backups
* Persisting all operations one after the other (memory to disk).
* That is if servers and redis collapses we still have all the data to restart redis.

→Note: its a good practice to not to store data in the same servers are running as per the following figure:



Redis on Flash:

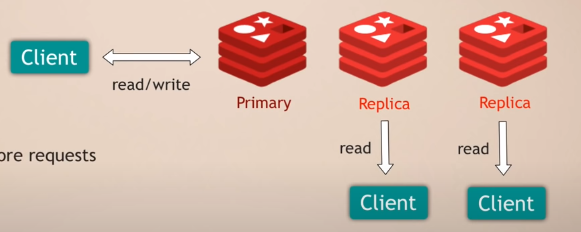
* Extends the RAM to the Hard drive:
* Hot values (frequently used) values are stored in RAM
* Warm values(infrequently used) values are stored on SSD
* Lower infrastructure cost.

Scaling Redis Database:

* Redis may face memory issues which can hinder the app performance (bottleneck).

1. Clustering:

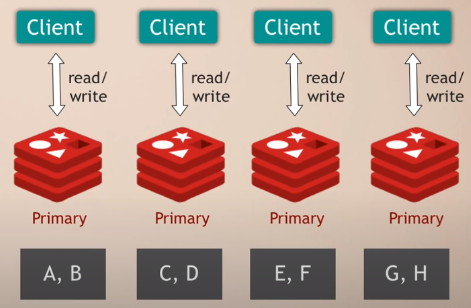
* Primary (or Master) for reading and writing.
* Multiple replicas for reading like:



* Replicas carry copies and occupy space.(not in same servers)
* If primary fails the replica becomes new master.

1. Sharding:

* Divide data.set into shards(smaller chunks).



* Less capacity.
* Scale onto smaller nodes.
* Re-shard as dataset grows.
* Performant and highly available database.
* But needs management better for teams working on maintaining rather than app development.

**Comparison with Rabbitmq:**

The table below summarizes the comparisons between RabbitMQ vs Redis:

|  |  |
| --- | --- |
| RabbitMQ | Redis |
| It is a dedicated message-broker. It is widely used in implementations of highly centralized and distributed systems. | It can function both as a message-broker, database, and cache. It is very fast and light-weight and hence grabs the attention of developers around the globe. |
| It can be scaled up to send around 50K messages per second. | It can be scaled up to send around a million messages per second. |
| It supports powerful message routing. It is useful when we need to run the same job on a specific server, group of servers or all servers. The application sends one message and the exchange will route is based on the routing key. | It supports the basic operations of a message-broker. For powerful message routing, people won’t go for Redis. |
| It can be easily configured to use SSL, thus providing an additional layer of security. | You have to opt for SSL separately which is a paid service. |
| It supports both persistent and transient messages. Developers would prefer RabbitMQ if they need to store messages for re-process in case of failures. | It supports only transient messages. Developers would prefer Redis over RabbitMQ for real-time Applications. |
| It inherently supported clustering and outperforms Redis in this aspect. | Clustering was introduced in Redis 3.0. |
| It supports both point to point and pub-sub messaging techniques. | It supports only the pub-sub mechanism. |
| RabbitMQ handles large messages way better than Redis. | It is better suited for smaller messages. The latency is very high for larger messages (>1 MB) |
| It guarantees delivery of messages and thus preferable when you can’t afford messages loss. | It doesn’t guarantee the delivery of each message. |

### Conclusion

It is very important to choose a message broker depending on your use case. As Redis provides extremely fast service and in-memory capabilities, you should prefer it for short retention of messages where persistence is not so important. On the other hand, you would prefer RabbitMQ when there is a requirement for complex routing.