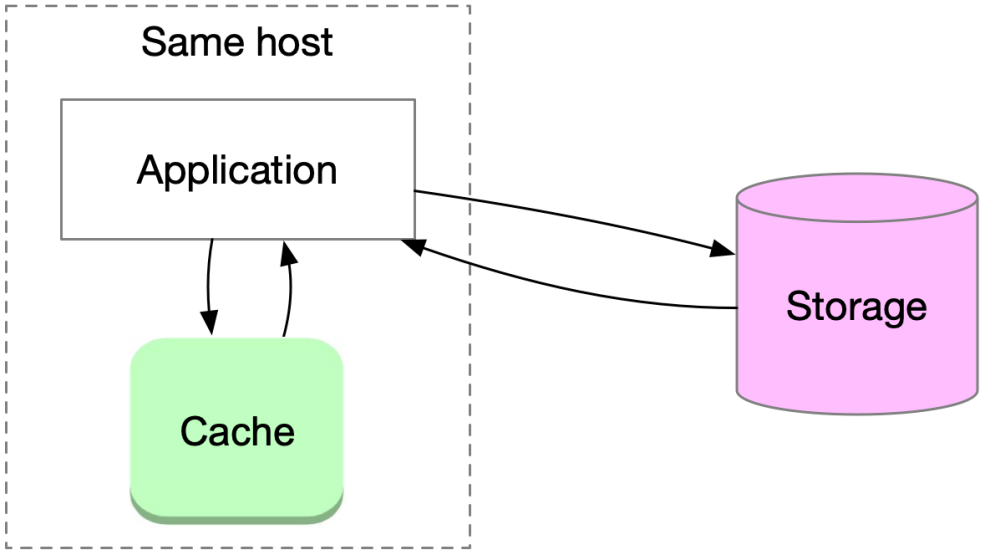
Cache

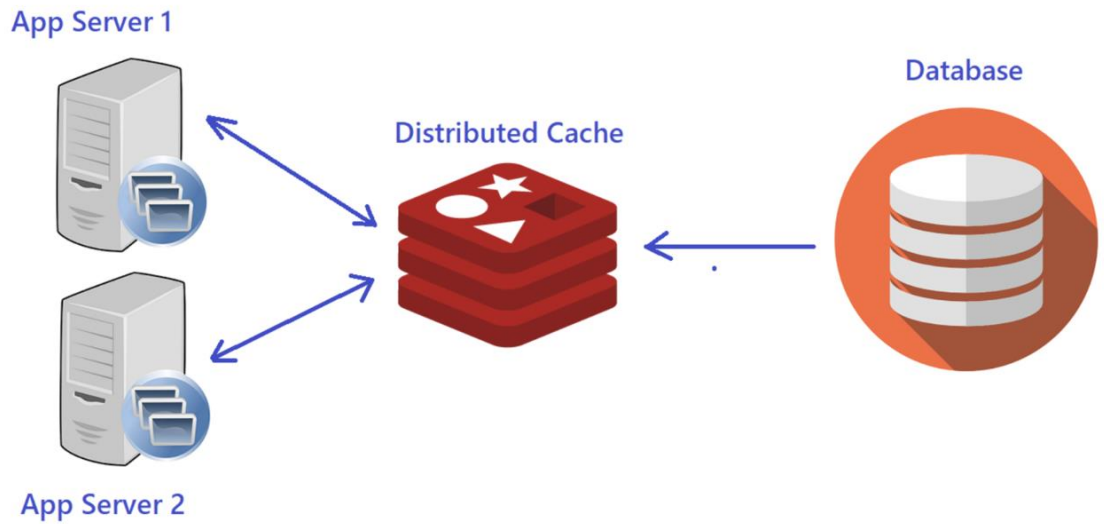
In process cache

- In-process cache is a type of cache that is located within the application itself
- It provides high read and write performance with zero network cost.
- However, it has limited data capacity that is restricted by the memory size, and cached data is lost if the process restarts.



in memory distributed cache

- A distributed cache is a system that pools together the random-access memory (RAM) of multiple networked computers into a single in-memory data store used as a data cache to provide fast access to data.



in memory distributed cache - Problems

- Data stored in RAM which is costly
- Performance do down, if all data stored in cache
- Only can store the subset of database

Eviction policy

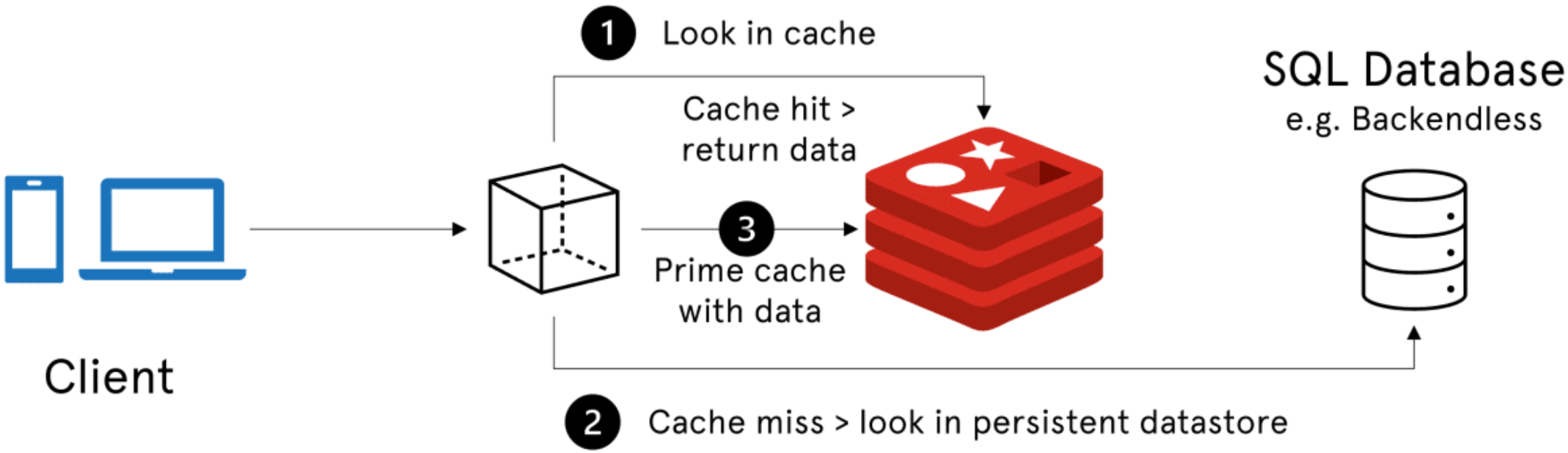
- The eviction policy determines what happens when a database reaches its memory limit.
- To make room for new data, older data is *evicted* (removed) according to the selected policy.

Redis

- Redis (**RE**mote **DI**ctionary **S**erver) is an open source, in-memory, NoSQL key/value store that is used primarily as an application cache or quick-response database.

- Redis stores data in memory, rather than on a disk or solid-state drive (SSD), which helps deliver unparalleled speed, reliability, and performance.

How Redis is typically used



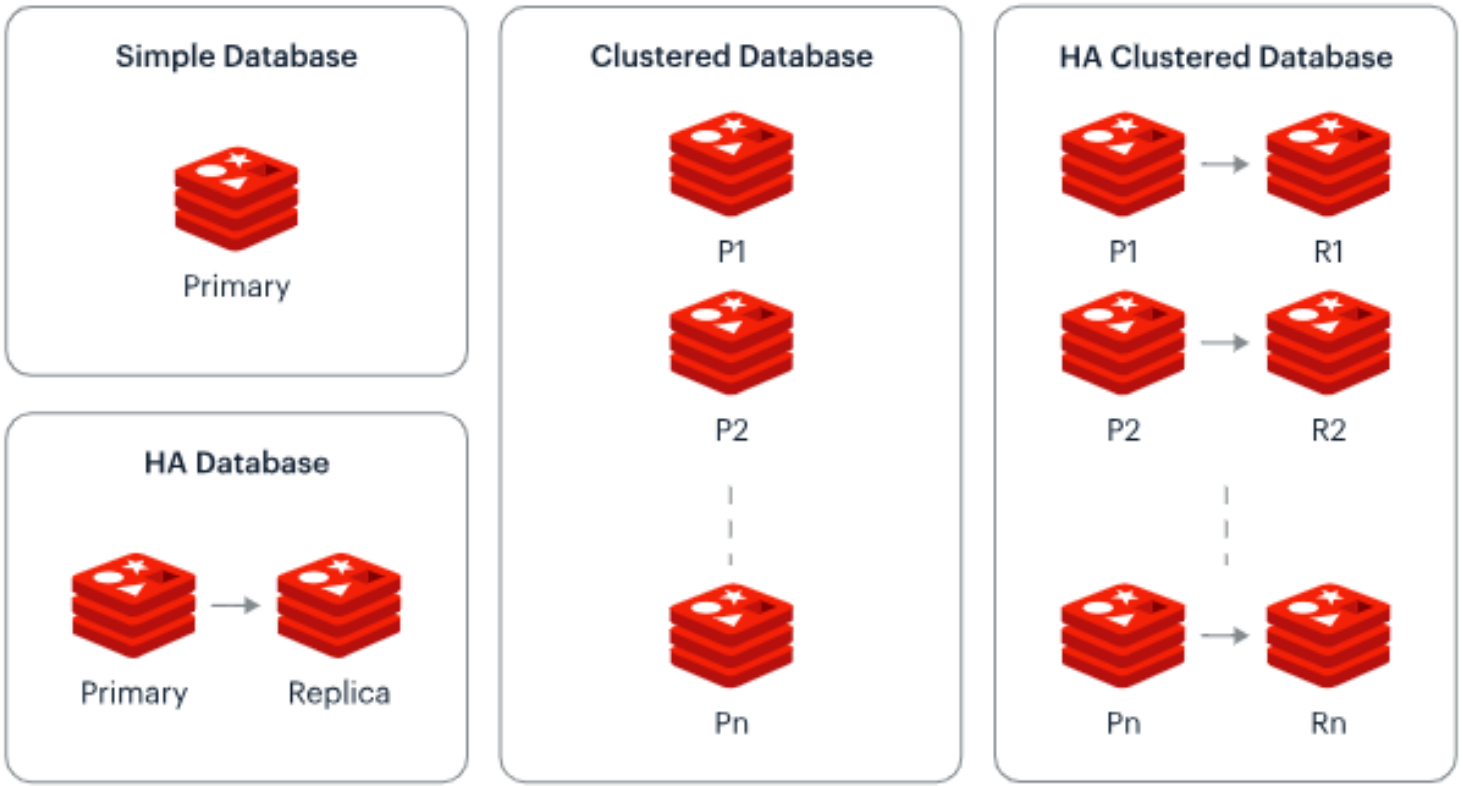
Redis Enterprise

- Redis Enterprise can be either a single Redis server database or a cluster

- Allows to either scale horizontally across many servers through sharding or to copy data, which ensures high availability with Redis Enterprise replicas

- Sharding is a type of database partitioning that separates large databases into smaller, faster, and more easily managed parts
- These smaller parts are called data shards.

CLUSTER ARCHITECTURE



Redis Replication

- Redis replication is a core feature that allows data from one Redis server (the master) to be copied to one or more other Redis servers (the slaves).

- This setup provides redundancy, load balancing, and can be used for read scaling, disaster recovery, and high availability.

Basic Master-Slave Replication

- **Master Node**: The primary Redis server where all write operations are directed. It holds the authoritative version of the data.
- **Slave Node(s) (Replica)**: Secondary Redis servers that replicate the data from the master. They are read-only by default and can serve read queries, which helps to distribute the load across multiple servers.

- Each master node can have **multiple replicas**

- Redis Enterprise cluster node can include between zero and a few hundred Redis databases in one of the following types:
- A simple database, i.e. a single primary shard
- A highly available (HA) database, i.e. a pair of primary and replica shards
- A clustered database, which contains multiple primary shards, each managing a subset of the dataset (or in Redis terms, a different range of “hash-slots”)
- An HA clustered database, i.e. multiple pairs of primary/replica shards

Sharding

- Sharding is a database architecture technique used to distribute data across multiple servers or nodes, allowing a system to scale horizontally.

- Instead of storing all the data on a single server, sharding divides the dataset into smaller, more manageable pieces called "shards."

How Sharding Works

- **Data Partitioning**: The key space (all possible keys) is divided into multiple ranges or slots, and each range/slot is assigned to a different shard.
- **Key Hashing**: Redis typically uses consistent hashing operation on the key to determine which shard a particular key belongs to
- **Routing Requests**: When a client sends a command to the cluster, the system determines the appropriate shard based on the key

Sharding

- Imagine you have 3 Redis nodes (Node A, Node B, and Node C). A simple modulo-based sharding strategy might assign keys as follows:
  - Keys with `hash(key) % 3 == 0` go to Node A.
  - Keys with `hash(key) % 3 == 1` go to Node B.
  - Keys with `hash(key) % 3 == 2` go to Node C.

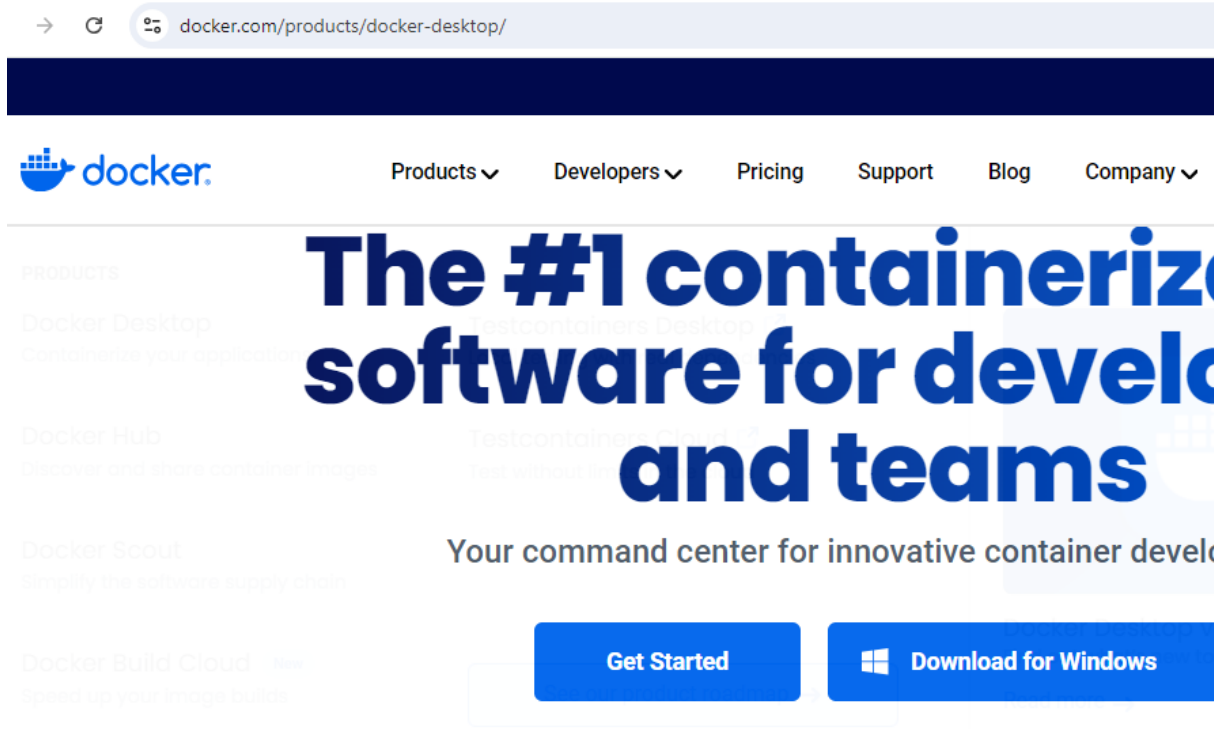
Eviction policy

Eviction Policy	Description
noeviction	New values aren't saved when memory limit is reached When a database uses replication, this applies to the primary database
allkeys-lru	Keeps most recently used keys; removes least recently used (LRU) keys
allkeys-lfu	Keeps frequently used keys; removes least frequently used (LFU) keys
allkeys-random	Randomly removes keys
volatile-lru	Removes least recently used keys with `expire` field set to true
volatile-lfu	Removes least frequently used keys with `expire` field set to true
volatile-random	Randomly removes keys with `expire` field set to true
volatile-ttl	Removes least frequently used keys with `expire` field set to true and the shortest remaining time-to-live (TTL) value

How to use redis in Windows

- Download docker desktop
- Restart the system
- Open the command prompt
- Docker pull redis

```
C:\Users\ALBIN XAVIER>docker pull redis
```



Running redis is docker

- **to run a image with port number 6379**
- docker run --name our-redis   -p 6379:6379 -d redis
- our-redis is container name

- To display all running containers
- docker ps

```
C:\Users\ALBIN XAVIER>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS          PORTS                    NAMES
990ff6b4415d   redis     "docker-entrypoint.s…"   43 hours ago   Up 16 seconds   0.0.0.0:6379->6379/tcp   our-r

C:\Users\ALBIN XAVIER>
```

To open redis shell

```
C:\Users\ALBIN XAVIER>docker exec -it 990 sh
#
```

Here 990 is the container id

```
C:\Users\ALBIN XAVIER>docker exec -it 990 sh
# redis-cli
127.0.0.1:6379> keys *
1) "Usersusers::102"
2) "\xac\xed\x00\x05t\x00\buser:102"
3) "Product"
4) "Usersusers::99"
127.0.0.1:6379>
```


THANK YOU!