



UT02: ALMACENAMIENTO DE DATOS

ÍNDICE

1.- Introducción al almacenamiento en Big Data

2.- Tipos de almacenamiento en Big Data

3.- Bases de datos clave-valor: Redis

4.- Bases de datos de documentos: MongoDB

5.- Bases de datos columnares: Cassandra

6.- Bases de datos de grafos: Neo4j

7.- Bases de datos de serie de tiempo: InfluxDB

X.- Data Warehouses vs Data Lakes

3.- Sistemas de almacenamiento en la nube

1

INTRODUCCIÓN AL ALMACENAMIENTO EN BIG DATA



1

INTRODUCCIÓN AL ALMACENAMIENTO EN BIG DATA

1.1

¿QUÉ ES EL ALMACENAMIENTO EN BIG DATA?

El almacenamiento en Big Data se refiere a las **arquitecturas, sistemas y tecnologías** diseñadas para **almacenar, gestionar y procesar volúmenes masivos de datos**.

Son cantidades tan grandes que los sistemas de bases de datos tradicionales no pueden manejarlos

Las soluciones comunes incluyen:



Almacenamiento en la **nube** (S3, Azure Blob Storage)



Sistemas de archivos **distribuidos** (Hadoop)



Bases de datos **NoSQL** para datos no estructurados

ESQUEMA



Esquema fijo y rígido (Schema-on-Write).

Se define la estructura de la tabla antes de insertar datos.

Esquema flexible o dinámico (Schema-on-Read).

La estructura se interpreta al leer los datos, permitiendo variedad.



ESCALABILIDAD



Escalabilidad vertical (Scale-Up)

Se aumenta la potencia (CPU, RAM, disco) de un único servidor.
Tiene límites físicos y es costosa

Escalabilidad horizontal (Scale-Out).

Se añaden más servidores commodity (económicos) al clúster. Es casi ilimitada y más económica.



ARQUITECTURA



Centralizada

Un único servidor gestiona todos los datos y operaciones.

Distribuida

Los datos y el procesamiento se reparten entre decenas, cientos o miles de nodos



TIPOS DE DATOS



Datos principalmente estructurados

Tablas con filas y columnas bien definidas

Todo tipo de datos

Estructurados, semi-estructurados (JSON, logs) y no estructurados (imágenes, videos).



MODELO DE DATOS



Relacional

Basado en tablas, claves primarias/foráneas y SQL

No relacional (NoSQL) o sistemas de archivos distribuidos

Optimizados para patrones de acceso específicos.



RENDIMIENTO



Optimizado para **transacciones complejas** (OLTP) y **consultas ACID**.

Optimizado para el procesamiento por lotes (batch) y, en algunos casos, tiempo real. Prioriza el *throughput* sobre la latencia en algunas operaciones.



COSTO



Alto

Suele utilizar hardware propietario y de alta gama, lo que incrementa el costo.

Bajo

Se basa en hardware commodity (estándar y económico) y software de código abierto, reduciendo costos.



2

BASES DE DATOS CLAVE-VALOR REDIS



2

BASES DE DATOS CLAVE-VALOR REDIS

2.1

¿QUÉ ES REDIS?

Redis (REmote DIctionary Server) es una base de datos en **memoria**, de tipo **clave-valor**, **no relacional (NoSQL)**.

Se caracteriza por su altísima velocidad, ya que guarda los datos principalmente en memoria RAM, aunque también permite la persistencia en disco.

Es software libre, desarrollado en C, y está disponible bajo la licencia BSD



Sus principales **características** son:

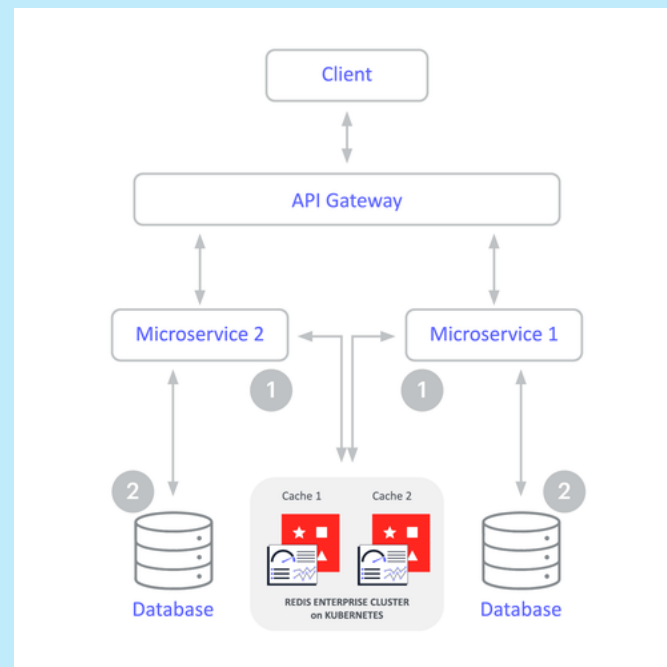
Almacenamiento en memoria	Estructuras de datos avanzadas	Persistencia opcional
Redis guarda los datos en RAM, lo que permite operaciones extremadamente rápidas	No solo almacena cadenas, sino también listas, conjuntos, hashes, conjuntos ordenados, streams, ...	Aunque funciona en memoria, puede guardar datos en disco mediante mecanismos como RDB o AOF.
Simplicidad y rendimiento	Alta disponibilidad y escalabilidad	Soporte para Pub/Sub
Su arquitectura es muy eficiente, basada en un solo hilo, lo que facilita la predicción del rendimiento	Soporta replicación maestro-esclavo, clústeres y Redis Sentinel para alta disponibilidad	Redis permite enviar y recibir mensajes entre procesos mediante un sistema de publicación y suscripción

Algunos ejemplos de **casos de uso** de Redis:

Caché de alto rendimiento

Se usa frecuentemente como **caché en memoria** para almacenar datos que deben consultarse rápidamente y que no cambian con frecuencia.

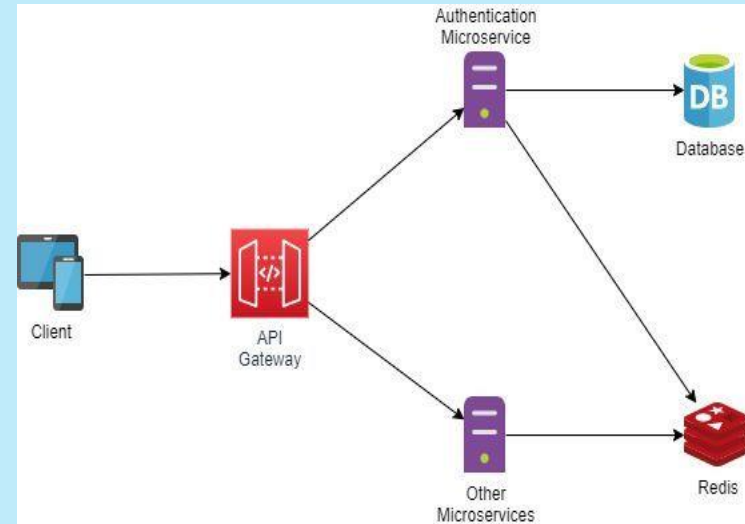
Esto reduce la carga del servidor principal y mejora significativamente el tiempo de respuesta de las aplicaciones web.



Almacenamiento de sesiones

Redis permite almacenar datos de sesión de usuarios, como tokens, ID de usuario, preferencias, ...

Compatible con múltiples *frameworks* y permite establecer tiempos de expiración automáticos.



Contadores y estadísticas tiempo real

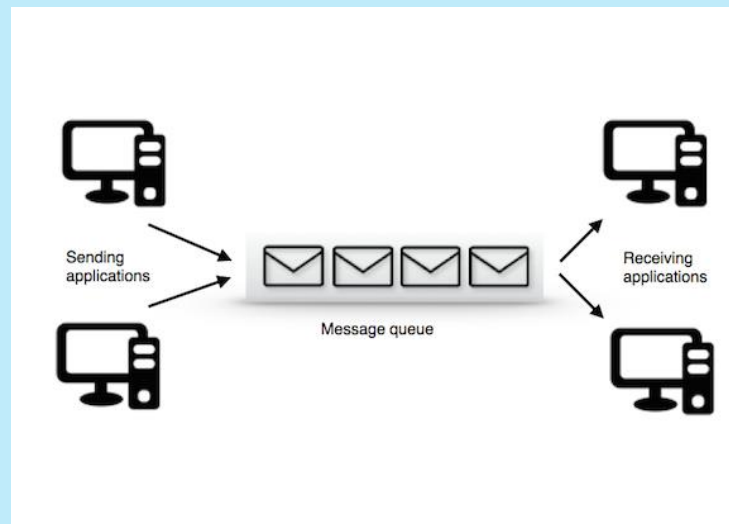
Gracias a sus operaciones atómicas y velocidad, Redis es ideal para implementar **contadores y métricas en tiempo real**.

Por ejemplo: visitas a una página, número de clicks de un botón o seguimiento de eventos en tiempo real (*likes*)



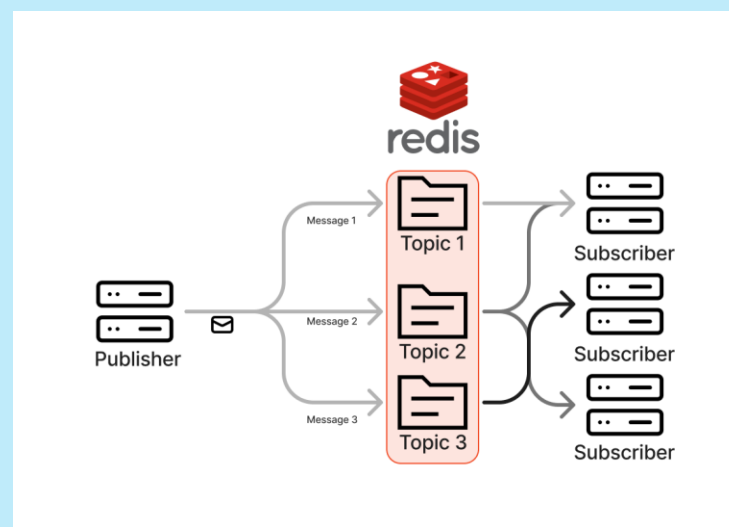
Colas de trabajo

Redis permite implementar **colas de tareas asíncronas** entre procesos o servicios. Esto es útil para **desacoplar procesos pesados** que no deben ejecutarse durante la respuesta al usuario: procesamiento de imágenes, e-mails, generación de PDFs...



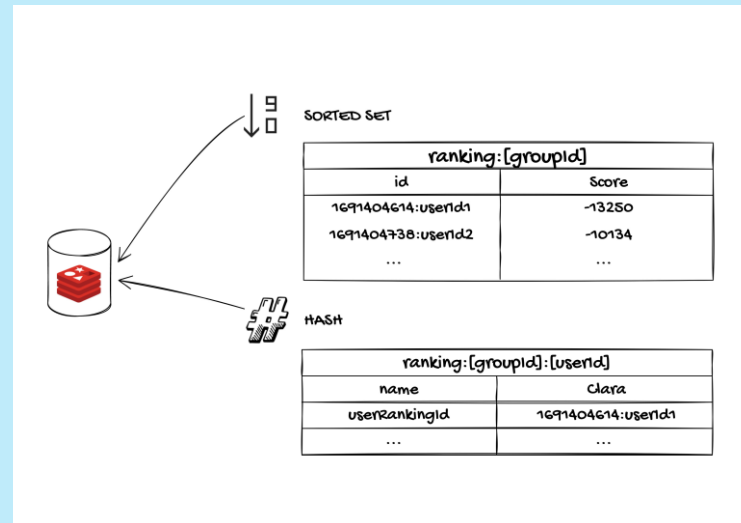
Sistemas de publicación/suscripción

Redis incluye un **sistema Pub/Sub** para permitir la comunicación entre procesos, útil en sistemas distribuidos o aplicaciones en tiempo real: chats, notificaciones a clientes, juegos multijugador,...



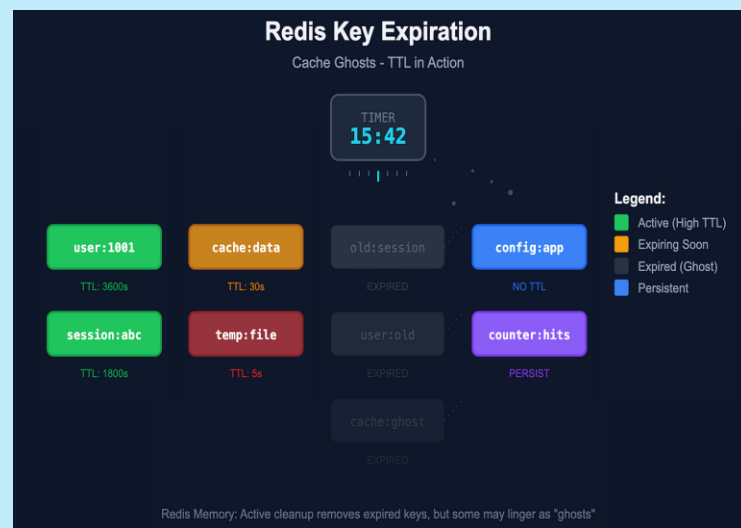
Rankings y puntuaciones

El tipo de datos **Sorted Set** permite mantener elementos ordenados por una puntuación, ideal para rankings (usuarios en un videojuego, puntuaciones en un concurso) o simplemente resultados ordenados por relevancia.



Almacenamiento temporal

Redis permite asignar **tiempos de vida (TTL)** a las claves, lo que es útil para almacenar datos que deben eliminarse automáticamente: tokens de autenticación, códigos temporales de verificación (OTP) o cachés de resultados que caducan.



2

BASES DE DATOS CLAVE-VALOR REDIS

2.2

INSTALACIÓN DE REDIS

docker-compose.yml

```
1  >Run All Services
2  services:
3    >Run Service
4    redis:
5      image: redis:7.2
6      container_name: redis-server
7      ports:
8        - "6379:6379"
9      volumes:
10       - redis-data:/data
11      command: ["redis-server", "--appendonly", "yes"]
12      restart: unless-stopped
13      networks:
14        - bigdata-network
15
16 volumes:
17   redis-data:
18
19 networks:
20   bigdata-network:
21     external: true
```

En este ejemplo he instalado la versión 7.2

Redis se expone en el puerto 6379

Si queremos persistencia debemos mapear el volumen /data

Recuerda que esta red es la que vamos a utilizar para interconectar todos nuestros contenedores

Acceso con Redis CLI

Una vez arrancado el contenedor comprobamos que funciona correctamente ejecutando **redis-cli** dentro del mismo

```
PS C:\> docker exec `
>> -it `
>> redis-server `
>> redis-cli
127.0.0.1:6379> █
```

Veamos ahora los comandos básicos de **administración general** de Redis.

PING

Verifica si el servidor Redis está activo, responde PONG si hay conexión con el servidor.

```
127.0.0.1:6379> PING
PONG
127.0.0.1:6379> █
```

QUIT

Cierra la conexión con Redis

INFO [sección]

Muestra información del servidor, opcionalmente se puede indicar una única sección (server, memory, ...)

```
127.0.0.1:6379> INFO memory
# Memory
used_memory:938704
used_memory_human:916.70K
used_memory_rss:14868480
used_memory_rss_human:14.18M
used_memory_peak:1158976
used_memory_peak_human:1.11M
used_memory_peak_perc:80.99%
used_memory_overhead:868056
used_memory_startup:865752
```

CONFIG GET <parámetro>

Consulta la configuración actual. Algunos parámetros son:

- `server_threads` → Hilos del servidor.
- `maxclients` → Número máximo de clientes conectados.
- `timeout` → Tiempo en segundos que se espera antes de cerrar conexiones inactivas.
- `tcp-keepalive` → Tiempo para mantener conexiones TCP vivas.
- `maxmemory` → Límite máximo de memoria (0 = sin límite).
- `maxmemory-policy` → Política de expulsión (noeviction, allkeys-lru, volatile-lru, etc.).
- `maxmemory-samples` → Número de muestras para algoritmos LRU/LFU.
- `hash-max-ziplist-entries` / `hash-max-ziplist-value` → Optimización de hashes pequeños.
- `bind` → Interfaces en las que escucha Redis.
- `port` → Puerto del servidor (por defecto 6379).
- `unixsocket` → Ruta de un socket UNIX (si se usa)
- `logfile` → Fichero de logs (si está configurado)

CONFIG GET <parámetro>

```
127.0.0.1:6379> config get maxclients
1) "maxclients"
2) "10000"
127.0.0.1:6379> config get logfile
1) "logfile"
2) ""
127.0.0.1:6379> config get maxmemory
1) "maxmemory"
2) "0"
```


CONFIG SET <parámetro> <valor>

Cambia los parámetros de configuración en tiempo real.

```
127.0.0.1:6379> config get maxclients
1) "maxclients"
2) "10000"
127.0.0.1:6379> config set maxclients 100
OK
127.0.0.1:6379> config get maxclients
1) "maxclients"
2) "100"
```

SELECT <índice>

Cambia a una base de datos diferente, identificando cada base de datos por un número entre 0 y 15

El uso de múltiples bases de datos no es común en producción, donde se suele utilizar **prefijos de claves** para manejar el aislamiento.

SELECT solo afecta a la conexión actual, si se abre una nueva conexión se empezará nuevamente en la base de datos 0

DBSIZE

Devuelve el número de claves en la base de datos actual

```
127.0.0.1:6379> DBSIZE
(integer) 3
127.0.0.1:6379> SELECT 1
OK
127.0.0.1:6379[1]> DBSIZE
(integer) 0
127.0.0.1:6379[1]>
127.0.0.1:6379[1]> █
```

FLUSHDB

Borra todas las claves de la base de datos actual

```
127.0.0.1:6379> DBSIZE  
(integer) 3  
127.0.0.1:6379> FLUSHDB  
OK  
127.0.0.1:6379> DBSIZE  
(integer) 0  
127.0.0.1:6379> █
```

FLUSHALL

Borra todas las claves de **todas** las bases de datos

CLIENT LIST

Muestra los clientes conectados

```
127.0.0.1:6379> client list  
id=25 addr=127.0.0.1:39940 laddr=127.0.0.1:6379 fd=8 name=  
lti-mem=0 rbs=1024 rbp=0 obl=0 oll=0 omem=0 tot-mem=22426  
127.0.0.1:6379> 
```

CLIENT KILL <id>

Cierra la conexión de un cliente

Acceso con Redis Insight

Si queremos un cliente gráfico podemos instalar **Redis Insight**

The screenshot shows the Redis Insight website at redis.io/insight/. The header includes the Redis logo, navigation links for "Redis for AI", "Products", "Resources", "Docs", and "Pricing", a search bar, and buttons for "Login", "Book a meeting", and "Try Redis". A banner at the top mentions "We're talking new releases & fast AI at Redis Released. Join us in your city. Register now".

The main content area features the "Redis Insight" title and the tagline "Build with the official Redis database tool—for free". Below this, it states "Build, debug, and visualize in just a few clicks with our free developer tool featuring our AI-powered assistant, advanced CLI, and intuitive GUI." There are buttons for "Download it for free" and "Read our docs".

A section titled "Redis Insight works across" shows logos for Linux, Microsoft, and Apple, followed by the text "and seamlessly supports all Redis deployments".

On the right side, there is a preview of the Redis Insight application interface. It shows a "Redis Databases" table with columns for "Database alias", "Host/port", "Connection type", and "Last". The table lists several databases, including "Create free Redis Cloud Database".

Overlaid on the application preview is a "Redis Copilot" chat window. It contains a conversation where the user asks "Hey Redis Copilot, what can you do for me?" and the assistant responds with a list of capabilities: "Explaining Redis concepts", "Exploring use cases and solutions", "Formulating Redis Engine commands", and "Querying your data and understanding the indexing schemes". The user then asks "Great! So how many models of kids bikes do I have in my DB?", and the assistant provides a Redis query: `FT.AGGREGATE idx:rep1_bicycle @age [1,4,6] @price [0,10000] @year [2017,2017] 1 model AS Count`.

Para descargarlo tendremos que introducir nuestros datos



Redis Insight

Download a powerful tool for visualizing and optimizing data in Redis

Operating system *

Windows

First name *

Víctor

Last name *

González

Business email *

vjgonzalezr@educa.jcyl.es

Company *

IES San Andrés

Phone *

987 84 63 15

Job function *

Other

Country *

Spain

Yes, I would like to receive email marketing messages from Redis.



By submitting this form, I am agreeing to the [Redis Insight License Terms](#) and the Redis [Privacy Policy](#).

Download

Aceptamos el EULA

EULA and Privacy Settings



To optimize your experience, Redis Insight uses third-party tools.

☐ Usage Data

Help improve Redis Insight by sharing anonymous usage data. This helps us understand feature usage and make the app better. By enabling this, you agree to our [Privacy Policy](#).

☒ Encrypt sensitive information

Select to encrypt sensitive information using system keychain. Otherwise, this information is stored locally in plain text, which may incur security risk.

Notifications

☐ Show notification

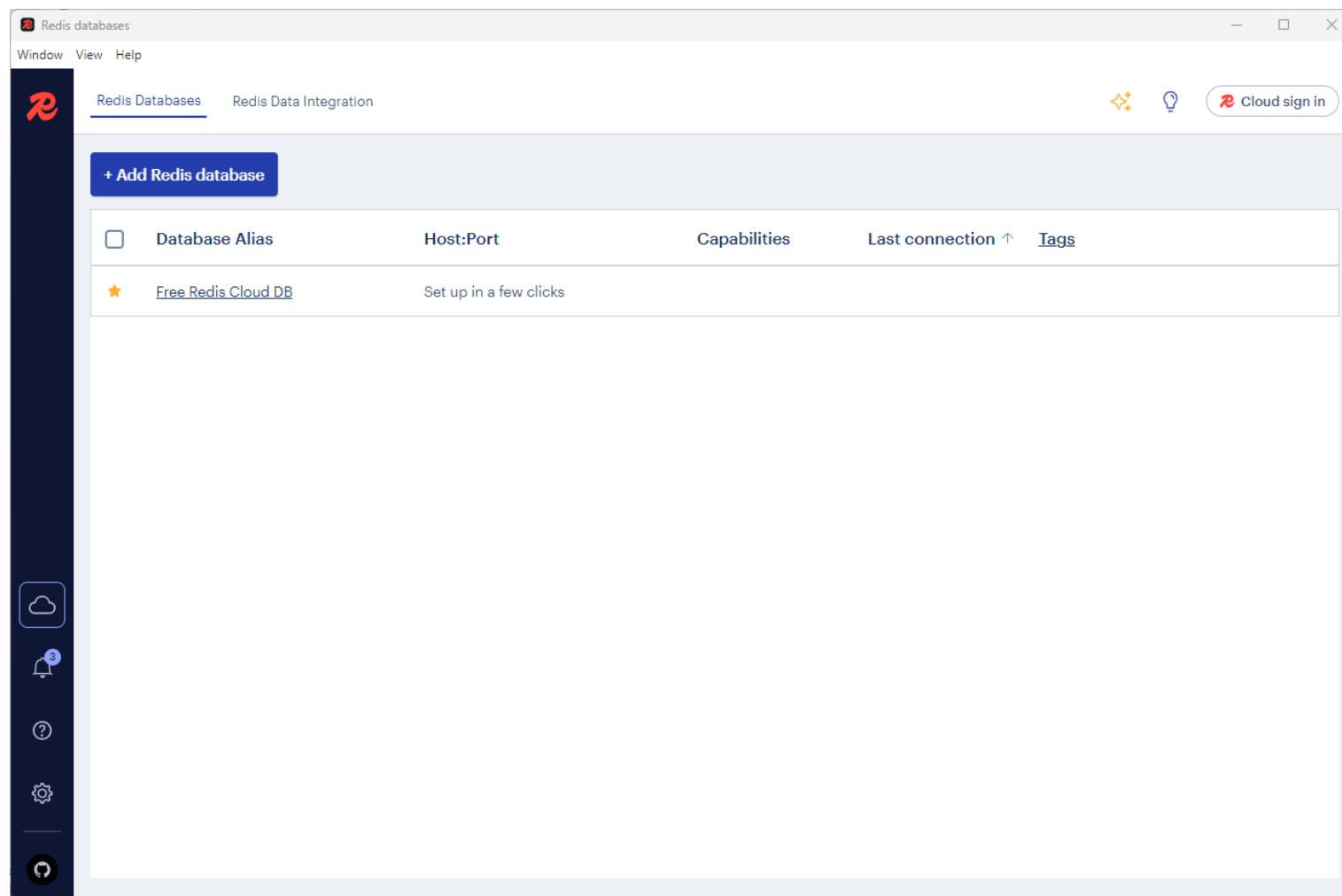
Select to display notification. Otherwise, notifications are shown in the Notification Center.

Use of Redis Insight is governed by your signed agreement with Redis, or, if none, by the [Redis Enterprise Software Subscription Agreement](#). If no agreement applies, use is subject to the [Server Side Public License](#)

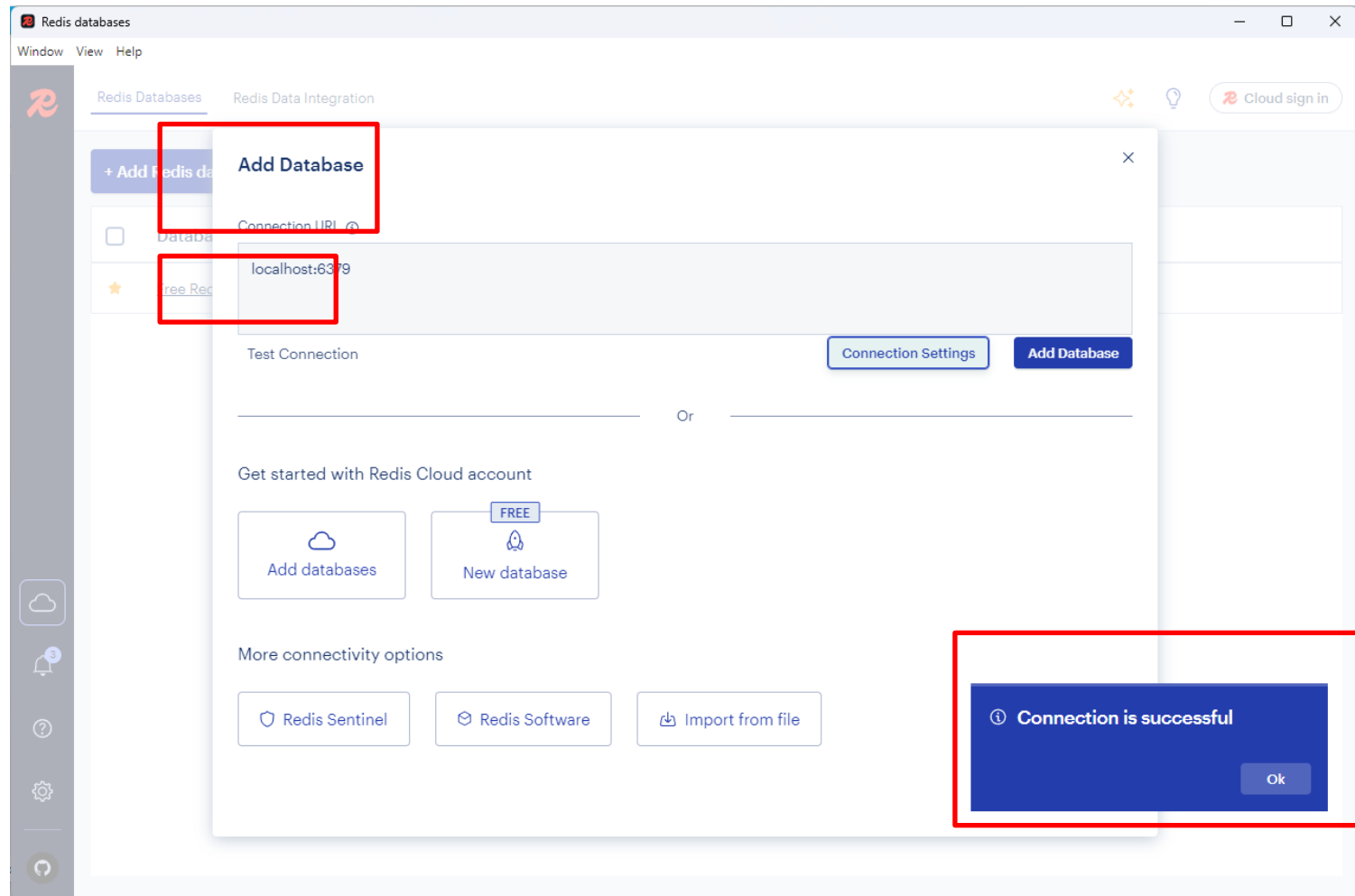
☒ I have read and understood the Terms

Submit

Y ya accedemos a la pantalla principal



El primer paso será conectarnos a una base de datos



Podremos tener conexiones a diferentes bases de datos

<div>+ Add Redis database</div>				
<div><div>Columns</div><div>Database List Search</div></div>				
<input type="checkbox"/>	Database Alias	Host:Port	Capabilities	Last connection ↑ Tags
<input checked="" type="checkbox"/>	Free Redis Cloud DB	Set up in a few clicks		
<input type="checkbox"/>	localhost:6379	localhost:6379		less than a minute ago

Acceso desde Python

Por último, también podremos acceder a Redis desde Python utilizando la librería `redis-py`

redis-py

The Python interface to the Redis key-value store.

 CI no status  docs  passing  license  MIT  pypi package  6.4.0  latest-prerelease  v7.0.0b1  codecov  93%

[Installation](#) | [Usage](#) | [Advanced Topics](#) | [Contributing](#)

NOTA: las siguientes capturas se realizan en un entorno configurado con un contenedor ejecutando **Redis** y otro con **Jupyter Notebook**.

```
PS C:\proyectos\docker_resources\docker_compose\jupyter_notebook> docker network create shared-network
a675558a092cd6dbed7c6f4aa5d4b1e3a362ebefc0ed535ed89bfad91c5dad1b
● PS C:\proyectos\docker_resources\docker_compose\jupyter_notebook> docker compose up -d
[+] Running 1/1
  ✓ Container jupyter_datascience Started
● PS C:\proyectos\docker_resources\docker_compose\jupyter_notebook> cd ../redis\
● PS C:\proyectos\docker_resources\docker_compose\redis> docker compose up -d
[+] Running 1/1
  ✓ Container redis-server Started
○ PS C:\proyectos\docker_resources\docker_compose\redis>
```

Recuerda que Redis se expone a través del puerto **6379** y Jupyter del **8888**. Asimismo, al estar conectados a la misma red externa (shared-network), podremos acceder a Redis desde Jupyter usando la resolución local de nombres de Docker.

<input type="checkbox"/>	<input checked="" type="checkbox"/>	jupyter_notebook	-	-	-
<input type="checkbox"/>	<input checked="" type="checkbox"/>	jupyter_datascience	a44d8dd78797	jupyter/datascience-noteb	8888:8888 ↗
<input type="checkbox"/>	<input checked="" type="checkbox"/>	redis	-	-	-
<input type="checkbox"/>	<input checked="" type="checkbox"/>	server	c5b180c89451	redis:7.2	6379:6379 ↗

El primer paso será instalar el módulo `redis-py` utilizando el comando `pip`

```
[1]: !pip install redis
```

```
Collecting redis
```

```
  Downloading redis-6.4.0-py3-none-any.whl.metadata (10 kB)
```

```
  Downloading redis-6.4.0-py3-none-any.whl (279 kB)
```

```
_____ 279.8/279.8 kB 4.4
```

```
Installing collected packages: redis
```

```
Successfully installed redis-6.4.0
```

Ten en cuenta que **pip** no es un comando Python, sino un comando del sistema para gestionar los paquetes instalados.

Para indicarle a Jupyter que se tiene que ejecutar en una terminal **lo precedemos del símbolo exclamación.**

Y ya podremos conectarnos

Importo el módulo
redis

```
import redis
```

Uso la clase
Redis que
representa la
conexión a la
base de
datos

```
r = redis.Redis(  
    host='redis',  
    port=6379,  
    db=0,  
    decode_responses=True  
)
```

Utilizo la resolución local de
nombres de Docker para
indicar el servicio que
contiene Redis

En este caso me estoy
conectando a la base de
datos 0

Con este parámetro
indicamos que devuelva
strings en lugar de bytes

```
print(r.ping())
```

True

Y pruebo la
conexión

Si no ha habido errores ya podré almacenar y recuperar valores en Redis

Uso la variable que me devolvió la conexión

```
r.set("nombre", "Victor")  
r.get("nombre")
```

El método `set` almacena un par clave-valor de tipo string, mientras que `get` lo recupera

'Victor'

Total: 1	2 min	Columns		STRING	nombre	64 B	Length: 6	TTL: No limit
STRING	nombre	No limit	64 B	Victor				

2

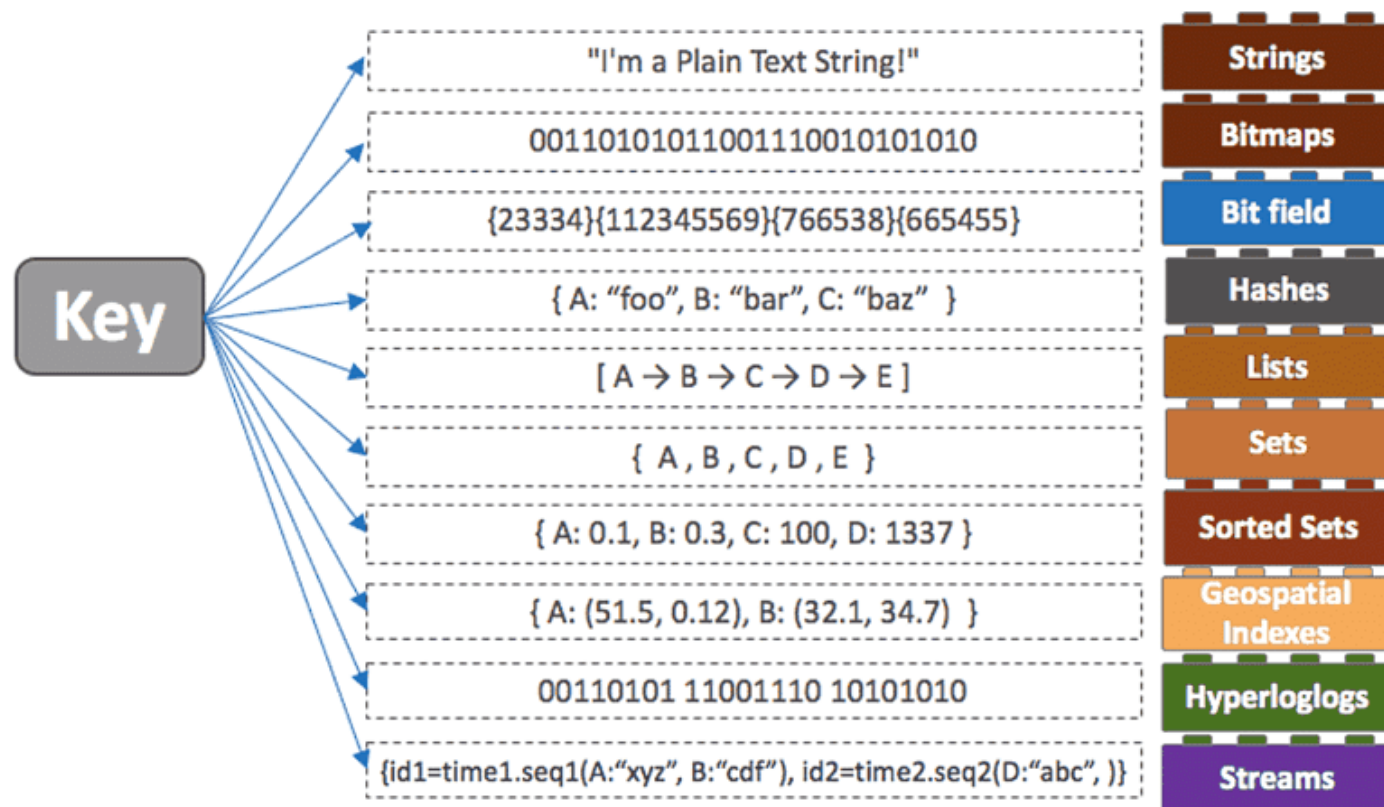
BASES DE DATOS
CLAVE-VALOR
REDIS

2.3

PRIMEROS PASOS CON
REDIS

Redis es una **base de datos clave-valor**, donde

- La **clave** siempre es una cadena
- El **valor** puede ser de diferentes tipos, según la información que queramos almacenar o el objetivo de la misma.



REDIS CLI SET <clave> <valor> / GET

Almacena **una clave y su valor** asociado en la base de datos. Guarda los valores en forma de **cadena de texto** (string)

Los valores se pueden recuperar con el comando **GET**

```
127.0.0.1:6379> SET nombre "Victor"  
OK  
127.0.0.1:6379> GET nombre  
"Victor"  
127.0.0.1:6379>
```

```
r.set("nombre", "Victor")  
r.get("nombre")
```

```
'Victor'
```

SET dispone de una serie de opciones adicionales:

- **EX**: establece un tiempo de expiración en segundos
- **PX**: igual que EX, pero en milisegundos
- **NX**: solo guarda la clave si no existe ya
- **XX**: solo guarda la clave si ya existe

```
127.0.0.1:6379> SET token "1234" EX 300  
OK  
127.0.0.1:6379> GET token  
"1234"
```

La clave se
eliminará
automáticamente
en 5 minutos

```
127.0.0.1:6379> SET nombre "Victor"  
OK  
127.0.0.1:6379> SET "nombre" "Victor Gonzalez" NX  
(nil)
```

Como la clave ya existe no escribe
nada en la base de datos

PYTHON

```
r.set(name, value, ex=None, px=None, nx=False, xx=False)
```

```
[7]: r.set("mensaje", "hola", ex=10)
```

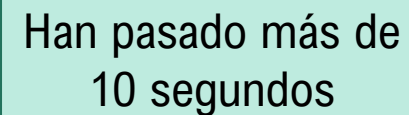
```
[7]: True
```

```
[8]: r.get("mensaje")
```

```
[8]: 'hola'
```

```
[9]: r.get("mensaje")
```

Han pasado más de
10 segundos



```
[ ]:
```

```
[10]: # Se guardará solo si 'contador' no existe  
r.set("contador", 1, nx=True)
```

```
[10]: True
```

```
[11]: r.get("contador")
```

```
[11]: '1'
```

```
[12]: r.set("contador", 100, nx=True)
```

Como la clave ya
existe no tiene
efecto

```
[13]: r.get("contador")
```

```
[13]: '1'
```

REDIS CLI KEYS <patrón>

Busca claves que coincidan con un patrón (ej: KEYS user:*).

No es conveniente usarlo en producción ya que es bloqueante.

```
127.0.0.1:6379> SET "user:name" "Victor"
OK
127.0.0.1:6379> SET "user:apellidos" "Gonzalez Rodriguez"
OK
127.0.0.1:6379> SET "pass" "1234"
OK
127.0.0.1:6379> KEYS user:*
1) "user:apellidos"
2) "user:name"
```

Observa que en Redis se suele usar una nomenclatura jerárquica con el carácter dos puntos como separador lógico.

El formato suele ser:
objeto:id:atributo

Por ejemplo:
usuario:100:apellidos

PYTHON

`r.keys (pattern="*")`

```
r.set("nombre", "Ana")  
r.set("curso", "Redis")  
r.set("contador", 5)
```

```
print(r.keys("c*"))
```

```
['contador', 'curso']
```


REDIS CLI EXISTS <clave>

Verifica si una clave existe

```
127.0.0.1:6379> EXISTS "user:name"  
(integer) 1  
127.0.0.1:6379> EXISTS "user:nickname"  
(integer) 0  
127.0.0.1:6379>
```

REDIS CLI DEL <clave>

Elimina una clave

```
127.0.0.1:6379> DEL "user:apellidos"  
(integer) 1  
127.0.0.1:6379> EXISTS "user:apellidos"  
(integer) 0  
127.0.0.1:6379>
```

PYTHON

```
r.exists (key)
```

PYTHON

```
r.delete (key1, key2, ...)
```

```
r.set("nombre", "Ana")

print(r.exists("nombre"))
print(r.exists("apellido"))
```

1

0

REDIS CLI EXPIRE <clave> <segundos>

Establece el tiempo de vida (TTL) de una clave en segundos

REDIS CLI TTL <clave>

Muestra el tiempo restante de vida de una clave (-2 si no tiene establecido tiempo de vida)

```
127.0.0.1:6379> EXPIRE "user:nombre" 60
(integer) 1
127.0.0.1:6379> TTL "user:nombre"
(integer) 52
127.0.0.1:6379> TTL "user:nombre"
(integer) 46
127.0.0.1:6379>
```

PYTHON

`r.expire(name, time)`

PYTHON

`r.ttl(name)`

PYTHON

`r.type(name)`

```
import time

r.set("mensaje", "Hola mundo", ex=10)
print("TTL inicial:", r.ttl("mensaje"))
time.sleep(3)
print("TTL después de 3s:", r.ttl("mensaje"))
time.sleep(8)
print("TTL después de 11s:", r.ttl("mensaje"))
```

Espera el tiempo indicado
como parámetro

```
TTL inicial: 10
TTL después de 3s: 7
TTL después de 11s: -2
```

Valores posibles:

- `>= 0`: segundos restantes
- `-1`: la clave existe, pero no tiene expiración
- `-2`: la clave no existe

REDIS CLI TYPE <clave>

Hasta ahora los valores solo han sido cadenas de texto, pero Redis soporta **múltiples tipos de datos** (que veremos en el siguiente apartado). Con el comando TYPE podemos ver qué tipo de datos almacena una determinada clave.

```
127.0.0.1:6379> TYPE "user:nombre"  
string  
127.0.0.1:6379>
```

2

BASES DE DATOS CLAVE-VALOR REDIS

2.4

TIPOS DE DATOS EN REDIS

CADENAS (STRINGS)

El tipo de dato más básico. Una **cadena** en Redis puede contener cualquier tipo de datos binarios (imágenes, texto serializado, etc.), hasta un tamaño de 512 MB.

Son ideales para almacenar valores simples como texto, números, o JSON.

Casos de uso comunes para las cadenas en Redis son:

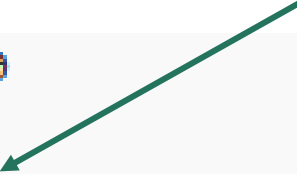
- **Caching**: almacenar resultados de consultas a bases de datos o APIs.
- **Contadores**: contadores de visitas, "*me gusta*", etc.
- **Almacenamiento de sesiones**: guardar datos de sesión de usuario serializados.
- **Banderas (flags)**: almacenar estados simples (ej. "activo", "inactivo").

Comandos para trabajar con cadenas:

- **SET *key value***: establece el valor de una clave
- **GET *key***: recupera el valor de una clave
- **DEL *key***: elimina una clave
- **INCR *key***: incrementa el valor numérico de una clave en 1
- **DECR *key***: decrementa el valor numérico de una clave en 1
- **EXPIRE *key seconds***: establece un tiempo de vida (TTL) para una clave
- **TTL *key***: obtiene el tiempo de vida restante de una clave



```
127.0.0.1:6379> SET user:1:name "Victor"  
OK  
127.0.0.1:6379> GET user:1:name  
"Victor"
```

Incrementamos el
valor en 1



```
127.0.0.1:6379> SET page:views 0  
OK  
127.0.0.1:6379> INCR page:views  
(integer) 1  
127.0.0.1:6379> GET page:views  
"1"
```

La clave se
eliminará
automáticamente
en 60 segundos



```
127.0.0.1:6379> SET user:token "1234" EX 60  
OK  
127.0.0.1:6379> TTL user:token  
(integer) 55
```

LISTAS (LISTS)

Las listas son una **colección ordenada** de cadenas.

Se implementan como listas enlazadas, lo que permite operaciones de **inserción y eliminación muy rápidas en los extremos**.

Son ideales para implementar **colas o pilas**.

Se utilizan habitualmente para:

- **Colas de mensajes:** implementar colas de tareas para procesadores en segundo plano.
- **Pilas (stacks):** historial de acciones recientes.
- **Feeds de redes sociales:** almacenar los últimos N elementos de un feed.
- **Listas de reproducción:** mantener el orden de elementos.

Comandos para trabajar con cadenas:

- **LPUSH *key value [value ...]***: añade uno o más valores al inicio de la lista.
- **RPUSh *key value [value ...]***: añade uno o más valores al final de la lista.
- **LPOP *key***: elimina y devuelve el primer elemento de la lista.
- **RPOP *key***: elimina y devuelve el último elemento de la lista.
- **LRANGE *key start stop***: obtiene un rango de elementos de la lista.
- **LLEN *key***: obtiene la longitud de la lista.
- **BLPOP *key [key ...] timeout***: versión bloqueante de LPOP, espera hasta que haya elementos.

```
127.0.0.1:6379> RPush tasks "task1" "task2" "task3"
```

Inicializamos una
lista con 3
elementos

```
(integer) 3
```

```
127.0.0.1:6379> LRange tasks 0 -1
```

```
1) "task1"
```

```
2) "task2"
```

```
3) "task3"
```

```
127.0.0.1:6379> LLen tasks
```

```
(integer) 3
```

```
127.0.0.1:6379> LPop tasks
```

```
"task1"
```

```
127.0.0.1:6379> LLen tasks
```

```
(integer) 2
```

```
127.0.0.1:6379> LRange tasks 0 -1
```

```
1) "task2"
```

```
2) "task3"
```

Con 0 -1 indicamos
desde el primer elemento
hasta el último

POP devuelve el primer
elemento de la lista y
también lo elimina

PYTHON

```
r.lpush(name, *values)
r.rpush(name, *values)
r.lpop(name, count=None)
r.rpop(name, count=None)
r.lrange (name, start, end)
r.llen(name)
r.blpop(keys, timeout=None)
```

```
lista_nombre = "tareas"

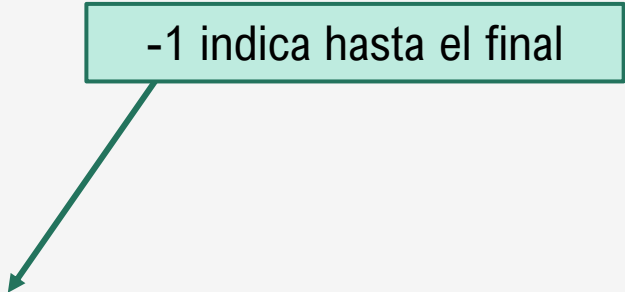
# Agregar elementos a la lista (al final)
r.rpush(lista_nombre, "Estudiar Python")
r.rpush(lista_nombre, "Hacer ejercicio")
r.rpush(lista_nombre, "Comprar comida")

# Mostrar todos los elementos de la lista
todas_tareas = r.lrange(lista_nombre, 0, -1)
print("Lista completa de tareas:", todas_tareas)

# Sacar un elemento desde el principio (FIFO)
tarea_inicial = r.lpop(lista_nombre)
print("Tarea eliminada desde el inicio:", tarea_inicial)

# Sacar un elemento desde el final (LIFO)
tarea_final = r.rpop(lista_nombre)
print("Tarea eliminada desde el final:", tarea_final)

# Mostrar la lista actualizada
tareas_actualizadas = r.lrange(lista_nombre, 0, -1)
print("Lista actualizada de tareas:", tareas_actualizadas)
```



-1 indica hasta el final

CONJUNTOS (SETS)

Un conjunto es una **colección desordenada de cadenas únicas**.

Los conjuntos son ideales para almacenar elementos donde **el orden no importa y se necesita asegurar la unicidad**, o para realizar operaciones de teoría de conjuntos.

Los conjuntos se suelen utilizar para:

- **Etiquetas (tags):** asociar tags a artículos, fotos, etc.
- **Miembros únicos:** lista de usuarios únicos en un chat, visitantes únicos.
- **Relaciones:** "usuarios que siguen a X", "productos en el carrito".
- **Operaciones de conjuntos:** amigos en común, usuarios con intereses compartidos

Los comandos para trabajar con conjuntos son:

- **SADD** *key member [member ...]*: añade uno o más miembros a un conjunto.
- **SREM** *key member [member ...]*: elimina uno o más miembros de un conjunto.
- **SMEMBERS** *key*: obtiene todos los miembros de un conjunto.
- **SISMEMBER** *key member*: comprueba si un miembro existe en un conjunto.
- **SCARD** *key*: obtiene el número de miembros en un conjunto.
- **SINTER** *key [key ...]*: devuelve la intersección de dos o más conjuntos.
- **SUNION** *key [key ...]*: Devuelve la unión de dos o más conjuntos.


```
127.0.0.1:6379> SADD users:online "Alice" "Bob" "Charlie"
(integer) 3
127.0.0.1:6379> SMEMBERS users:online
1) "Alice"
2) "Bob"
3) "Charlie"
127.0.0.1:6379> SISMEMBER users:online "Alice"
(integer) 1
127.0.0.1:6379> SISMEMBER users:online "Edgar"
(integer) 0
127.0.0.1:6379> SADD users:premium "Alice" "David"
(integer) 2
127.0.0.1:6379> SINTER users:online users:premium
1) "Alice"
```

Muestra
todos los
valores del
conjunto

Si contiene la
clave devuelve
1 y 0 en caso
contrario

Obtenemos la
intersección de
dos conjuntos

PYTHON

```
r.sadd(name, *values)
r.srem(name, *values)
r.smembers(name)
r.sismember(name, value)
r.scard (name)
r.sinter(keys, *args)
r.sunion(keys, *args)
```


CONJUNTOS ORDENADOS (SORTED SETS)

Son similares a los conjuntos, pero cada miembro único tiene un **score** (puntuación) asociado, que es un valor numérico de coma flotante.

Los elementos se mantienen ordenados por su score. Si varios miembros tienen el mismo score, se ordenan lexicográficamente.

Su uso más común es para:

- **Tablas de clasificación (leaderboards):** rankings de juegos, puntuaciones.
- **Clasificación de elementos:** artículos más populares por votos, usuarios más activos.
- **Series temporales simples:** eventos con un *timestamp* como score.

Se puede trabajar con conjuntos ordenados con los siguientes comandos:

- **ZADD *key score member [score member ...]***: añade uno o más miembros a un conjunto ordenado con su score.
- **ZRANGE *key start stop [WITHSCORES]***: obtiene un rango de miembros por índice (ordenado por score).
- **ZREVRANGE *key start stop [WITHSCORES]***: obtiene un rango de miembros en orden descendente.
- **ZRANGEBYSCORE *key min max [WITHSCORES]***: obtiene miembros dentro de un rango de scores.
- **ZREM *key member [member ...]***: elimina uno o más miembros.
- **ZSCORE *key member***: obtiene el score de un miembro.
- **ZCARD *key***: obtiene el número de miembros.

```
127.0.0.1:6379> ZADD leaderboard 100 "Alice" 85 "Bob" 120 "Charlie"  
(integer) 3
```

```
127.0.0.1:6379> ZRANGE leaderboard 0 -1 WITHSCORES
```

```
1) "Bob"  
2) "85"  
3) "Alice"  
4) "100"  
5) "Charlie"  
6) "120"
```

Obtenemos todas las
claves ordenadas por
puntuación

```
127.0.0.1:6379> ZREVRANGE leaderboard 0 -1 WITHSCORES
```

```
1) "Charlie"  
2) "120"  
3) "Alice"  
4) "100"  
5) "Bob"  
6) "85"
```

Ahora ordenado de mayor
a menor

```
127.0.0.1:6379> ZSCORE leaderboard "Alice"  
"100"
```

Puntuación asociada a un
valor

PYTHON

```
r.zadd(name, mapping, nx=False, xx=False, ch=False,
incr=False)
```

```
r.zrange(name, start, end, desc=False, withscores=False,
score_cast_func=Float)
```

```
r.zrevrange(name, start, end, withscores=False,
score_cast_func=float)
```

```
r.zrangebyscore (name, min, max, start=None, num=None,
withscores=False, score_cast_func=float)
```

```
r.zrem (name, *values)
```

```
r.zscore(name, value)
```

```
r.zcard(name)
```

HASHES (HASH MAPS)

Un hash es un **mapa de campos a valores**.

Cada hash puede almacenar hasta $2^{32} - 1$ pares campo-valor.

Son perfectos para representar objetos o registros de datos donde cada campo tiene un nombre y un valor asociado.

Los hashes se suelen utilizar para:

- **Almacenamiento de objetos:** perfiles de usuario, detalles de productos.
- **Caché de objetos:** almacenar objetos complejos de forma estructurada.
- **Configuraciones:** guardar configuraciones de aplicaciones o usuarios.

Redis dispone de los siguientes comandos para trabajar con hashes:

- **HSET** *key field value [field value ...]*: establece el valor de uno o más campos en un hash.
- **HGET** *key field*: obtiene el valor de un campo específico.
- **HGETALL** *key*: obtiene todos los campos y valores de un hash.
- **HDEL** *key field [field ...]*: elimina uno o más campos de un hash.
- **HKEYS** *key*: obtiene todos los nombres de campo en un hash.
- **HVALS** *key*: obtiene todos los valores en un hash.
- **HLEN** *key*: obtiene el número de campos en un hash

PYTHON

```
r.hset(name, key=none, value=none, mapping=none)
```

```
r.hget (name, key)
```

```
r.hgetall (name)
```

```
r.hdel (name, *keys)
```

```
r.hkeys (name)
```

```
r.hvals (name)
```

```
r.hlen(name)
```

STREAMS

Son un tipo de dato de **solo anexar** (*append-only*) que representa un log de eventos.

Cada entrada en un *stream* tiene un ID único y un conjunto de pares campo-valor.

Los *streams* están diseñados para sistemas de procesamiento de eventos y colas de mensajes persistentes.

Algunos casos de uso comunes son:

- **Logs de eventos:** almacenar un historial inmutable de eventos.
- **Sistemas de mensajería:** colas de mensajes persistentes con procesamiento distribuido (grupos de consumidores).
- **Procesamiento de datos en tiempo real:** ingesta y análisis de datos de sensores o IoT

Los principales comandos para trabajar con streams son:

- **XADD** *key ID field value [field value ...]*: añade una nueva entrada al stream. El ID puede ser * para generar uno automáticamente.
- **XRANGE** *key start end*: lee un rango de entradas de un stream.
- **XREAD** *[COUNT count] [BLOCK milliseconds] STREAMS key [key ...] ID [ID ...]*: lee entradas de uno o más streams, opcionalmente bloqueando hasta que haya nuevas entradas.
- **XGROUP CREATE** *key groupname ID [MKSTREAM]*: crea un grupo de consumidores para un stream.
- **XREADGROUP GROUP** *groupname consumername [COUNT count] [BLOCK milliseconds] STREAMS key [key ...] ID*: lee entradas de un stream usando un grupo de consumidores.

PYTHON

```
r.xadd(name, fields, id='*', maxlen=None,  
approximate=True, nomkstream=False)
```

```
r.xrange (name, min='-', max='+', count=None)
```

```
r.xread (streams, count=None, block=None)
```

```
r.xgroup_create (name, groupname, id='$', mkstream=False)
```

```
r.xgroup_destroy (name, groupname)
```

```
r.xgroup_delconsumer (name, groupname, consumername)
```

```
r.xreadgroup(groupname, consumername, streams, count=None,  
block=None, noack=False)
```

BITMAPS

Son un tipo de dato que permite tratar las cadenas de Redis como un **array de bits**.

Puedes establecer o borrar bits individuales en un offset específico.

Son extremadamente eficientes para almacenar información booleana a gran escala.

Los bitmaps se usan principalmente para:

- **Presencia de usuarios:** saber qué usuarios están online (cada offset es un ID de usuario).
- **Flags de características:** marcar si un usuario ha usado una característica específica.
- **Análisis de cohortes:** identificar usuarios que realizaron una acción en un día específico.
- **Votaciones binarias:** registrar votos "sí/no".

Podemos trabajar con bitmaps con los siguientes comandos:

- **SETBIT** *key offset value*: establece o borra el bit en un offset dado.
- **GETBIT** *key offset*: obtiene el valor del bit en un offset dado.
- **BITCOUNT** *key [start end]*: cuenta el número de bits establecidos (a 1) en una cadena.
- **BITPOS** *key bit [start] [end]*: encuentra la posición del primer bit establecido o no establecido

PYTHON

```
r.setbit(name, offset, value)
```

```
r.getbit (name, offset)
```

```
r.bitcount (name, start=None, end=None)
```

```
r.bitpos (name, bit, start=None, end=None)
```


HYPERLOGLOG

Es un algoritmo **probabilístico** para estimar la cardinalidad de un conjunto (el número de elementos únicos) con un uso de memoria muy bajo.

Es ideal cuando necesitas **contar elementos únicos** en grandes volúmenes de datos y estás dispuesto a aceptar un **pequeño margen de error**.

Se usa habitualmente para:

- **Contadores de visitantes únicos:** en sitios web o aplicaciones.
- **Estimación de elementos únicos:** IPs únicas, IDs de productos vistos.
- **Análisis de tendencias:** ver la cardinalidad de eventos a lo largo del tiempo.

Dispone de los siguientes comandos:

- **PFADD** *key element [element ...]*: añade elementos a un *HyperLogLog*.
- **PFCOUNT** *key [key ...]*: estima la cardinalidad de uno o más *HyperLogLogs*.
- **PFMERGE** *destkey sourcekey [sourcekey ...]*: combina múltiples *HyperLogLogs* en uno.

PYTHON

```
r.pfadd(name, *values)
```

```
r.pdcount (*sources)
```

```
r.pfmerge (dest, *sources)
```

GEO-SPATIAL (Datos Geoespaciales)

Permite almacenar **coordenadas de latitud y longitud** y realizar **consultas basadas en la distancia**.

Utiliza conjuntos ordenados internamente para almacenar los puntos geoespaciales, lo que permite operaciones eficientes de búsqueda por radio o por caja.

Se usa habitualmente para:

- **Contadores de visitantes únicos:** en sitios web o aplicaciones.
- **Estimación de elementos únicos:** IPs únicas, IDs de productos vistos.
- **Análisis de tendencias:** ver la cardinalidad de eventos a lo largo del tiempo.

Los comandos de que disponemos son:

- **GEOADD** *key longitude latitude member [longitude latitude member ...]*: añade uno o más puntos geoespaciales.
- **GEODIST** *key member1 member2 [unit]*: calcula la distancia entre dos miembros.
- **GEORADIUS** *key longitude latitude radius unit [WITHCOORD] [WITHDIST] [WITHHASH] [COUNT count] [ASC|DESC]*: encuentra miembros dentro de un radio dado de un punto central.
- **GEORADIUSBYMEMBER** *key member radius unit [WITHCOORD] [WITHDIST] [WITHHASH] [COUNT count] [ASC|DESC]*: similar a GEORADIUS, pero centrado en un miembro existente.
- **GEOHASH** *key member [member ...]*: devuelve el geohash de uno o más miembros.

PYTHON

```
r.geoadd(name, mapping, nx=False, xx=False, ch=False)
```

```
r.geodist (name, member1, member2, unit='m')
```

```
r.georadius (streams, longitude, latitude, radius,  
unit='m', withdist=False, withcoord=False, withhash=False,  
count=None, store=None, store_dist=None)
```

```
r.georadiusbymember(name, member, radius, unit='m',  
withdist=False, withcoord=False, withhash=False,  
count=None, sort=None, store=None, store_dist=None)
```

```
r.geohash (name, *members)
```

3

BASE DE DATOS DE DOCUMENTOS: MONGODB

