

# Los sistemas de caché: claves para un rendimiento superior

Backend



# Los sistemas de caché: claves para un rendimiento superior

Los sistemas de caché representan una de las estrategias más eficientes para mejorar la velocidad y eficiencia de las aplicaciones web. Un sistema de caché no es más que una capa de almacenamiento temporal que guarda copias de datos o archivos en una localización que permite un acceso rápido.

En un mundo en el que la inmediatez es primordial, estos sistemas son esenciales para la experiencia del usuario y el rendimiento de las aplicaciones. A lo largo de este fastbook entenderás el porqué.

*Autor: Antonio Blanco Oliva*

- Introducción a los sistemas de caché
- Tipos de caché en aplicaciones web
- El sistema Redis
- Publicación y suscripción en Redis (Pub/Sub)
- Gestión de sesiones con Redis
- Interacciones de aplicaciones y caché
- Conclusiones

# Introducción a los sistemas de caché



Las **ventajas** que nos proporciona el uso de caché son múltiples, pero me gustaría destacar estas tres:

## La reducción de la latencia

Al almacenar datos en caché, se reduce significativamente el tiempo de acceso a estos, ya que se evita la necesidad de recuperarlos de fuentes más lentas, como bases de datos o servicios externos.

## La disminución de la carga del servidor

Al servir datos desde el caché, se reduce la carga en los servidores de bases de datos y otros sistemas backend, lo que es crucial durante picos de tráfico.

## La mejora de la experiencia del usuario

Una aplicación más rápida directamente mejora la experiencia del usuario, lo que puede traducirse en mayor retención y satisfacción del cliente.

También resulta primordial entender la naturaleza de los datos para aplicar caché. ¿Y qué tipos de datos podemos encontrarnos?

#### DATOS ESTÁTICOS

#### DATOS DINÁMICOS

Incluyen imágenes, archivos CSS y JavaScript, que rara vez cambian.

#### DATOS ESTÁTICOS

#### DATOS DINÁMICOS

Incluyen resultados de consultas a bases de datos o respuestas de APIs que, aunque pueden cambiar, suelen mantenerse constantes durante períodos determinados.

## Aspectos que debemos tener en cuenta

Cuando aplicamos un **sistema intermedio de caché**, son muchos los aspectos para tener en cuenta, por ejemplo:

- Las **políticas de expiración**. Hay que definir cuándo se debe actualizar o eliminar el contenido del caché.
- El **tamaño del caché**. La caché no es infinita, por lo que tendremos que definir el tamaño de esta.
- La **consistencia de datos**. Debemos asegurar que los datos en caché reflejan con precisión el estado actual de los datos, ya que, al ser fotos fijas de ellos, pueden quedar desincronizados con los reales.
- Saber qué cachear**. Podemos cachear una página completa, solo las imágenes, los datos de usuario... Por lo tanto, es necesario decidir qué cachear y qué no.

# Tipos de caché en aplicaciones web



El *caching* es una técnica esencial para mejorar el rendimiento de las aplicaciones web.

Como ya hemos adelantado, existen diversos tipos de caché y, si nuestro objetivo es optimizar determinados aspectos de la experiencia del usuario y/o de la carga del servidor, utilizaremos un tipo de caché u otro.

Por lo tanto, antes de avanzar, vamos a detenernos en las **tipologías más utilizadas** en las aplicaciones web.

1

## Caché de navegador

Almacena **recursos estáticos** (como imágenes, CSS, JavaScript...) **en el navegador del usuario**. Cuando el usuario vuelve a visitar la página, estos recursos se cargan desde el caché local en lugar de descargarlos nuevamente desde el servidor.

De esta manera, mejoramos no solo la velocidad de carga, sino también el consumo de ancho de banda, de ahí que sea un tipo de caché muy extendido en cuanto a su uso.

---

**Eso sí, hay que tener especial cuidado con la gestión de los archivos cacheados, versionándolos correctamente para que el usuario pueda tener siempre las últimas versiones disponibles de ellos.**

2

### **Redes de distribución de contenidos (CDN)**

---

Un CDN almacena copias de contenido estático en servidores distribuidos geográficamente.

Los usuarios acceden al contenido desde el servidor más cercano a su ubicación. Al tener los servidores a menor distancia de los destinos, se reducen los tiempos de carga y, además, nos permite disponer de un sistema de replicado en caso de errores.



**Recuerda:** el replicado correcto de la información en los distintos nodos es uno de los puntos que debemos tener en cuenta en su diseño.

3

### **Caché de páginas**

Almacena **versiones completas de páginas web**, es decir, cuando un usuario solicita una página, se sirve la versión cacheada en lugar de generarla dinámicamente. Gracias a esto, se logra que bajemos tanto los tiempos de carga de la página como la carga en el servidor web, ya que no tiene que reconstruirlas con cada petición de usuario.

Claro que, si son páginas muy dinámicas, hay que controlar mucho los tiempos de refresco, optando, en ocasiones, por políticas que dictamen que no se cachee la página para aquellos usuarios logueados.

4

#### Caché de memoria

Utiliza **la memoria RAM del servidor para almacenar datos** frecuentemente solicitados, como las sesiones de usuario o los resultados de consultas de base de datos.

Nos ofrece, por tanto, unos **accesos extremadamente rápidos a los datos**, pero al ser un sistema que trabaja en memoria tiene sus limitaciones de tamaño. Eso supone que haya que definir muy bien los tiempos de refresco.

5

#### Caché de objetos

Almacena objetos de datos, como las entidades de base de datos o los resultados de cálculos complejos y, por lo tanto, **evita la repetición de operaciones costosas** en cuanto a la generación de los datos.

Como ya puedes suponer, cada uno de los tipos de caché que acabamos de ver tiene sus propias ventajas y casos de uso ideales.

---

**La elección del tipo de caché más adecuado  
dependerá de los requisitos específicos de la  
aplicación, de la infraestructura disponible y el patrón  
de acceso a los datos.**

---

Por lo tanto, es fundamental **analizar estos tres elementos**. Y es que **implementación efectiva del caché** repercutirá directamente en **mejoras significativas** tanto en el rendimiento como en la escalabilidad de las aplicaciones web.

# El sistema Redis



Redis, que significa *Remote Dictionary Server*, es una base de datos no relacional en memoria, utilizada como sistema de caché y almacén de mensajes. Es conocido por su **rapidez y versatilidad**, cualidades que lo hacen ideal para una variedad de casos de uso en aplicaciones web modernas. ¿Y cuáles son sus bondades principales?

En primer lugar, al ser una base de datos NoSQL, no se basa en tablas para el almacenaje de los datos, sino que usa el modelo *clave:valor* y, por tanto, es capaz de almacenar más que datos sueltos, también listas, conjuntos, mapas...

En segundo lugar, su almacenamiento en memoria permite tiempos de lectura/escritura altamente rápidos en comparación con los sistemas de bases de datos basados en almacenamiento en disco. Además, que trabaje a nivel de memoria no impide que tenga mecanismos para persistir los datos, como el volcado en disco duro y la replicación, por tanto, ofrece una buena resistencia a fallos.

Por último, cabe destacar que soporta el sistema de sincronización de mensajes mediante publicadores/subscriptores (Pub/Sub). ¿Qué significa esto? Que Redis resulta muy práctico para el uso de aplicaciones de mensajería en tiempo real.

Veamos a continuación otros detalles de Redis de suma importancia para nuestro trabajo de desarrolladores.

## Estructuras de datos

Como ya hemos mencionado, Redis es conocido por su **versatilidad y eficiencia** en el manejo de diferentes tipos de estructuras de datos. Estudiemos las más relevantes.

### Strings

Hablamos de strings para referirnos a lo que solemos llamar 'los pares clave:valor'. Son el tipo de dato más simple en Redis, pero muy útiles para almacenar valores sencillos como los nombres de usuarios, las puntuaciones, etc.

### Listas

Aquí nos referimos a las listas ordenadas de strings, ideales para implementar colas, pilas o listas de seguimiento.

## Conjuntos

---

Estos son las colecciones no ordenadas de strings únicos, muy útiles para almacenar los elementos únicos y para realizar operaciones de conjuntos como la unión, la intersección y la diferencia.

## Hashes

---

Son las colecciones de pares clave-valor que se asignan o consultan a través de una determinada clave (*hash*). Los hashes son ideales para representar objetos (como usuarios con múltiples campos).

---

**Existen más tipos de estructura (bitmaps, streams...), pero nos centraremos en las primeras, ya que se usan con mucha frecuencia y serán las que usaremos en nuestros ejercicios prácticos para esta asignatura.**

## Casos de uso

Podemos trabajar con Redis en prácticamente cualquier tipo de aplicación, como sistema de caché en memoria para mejorar los tiempos de acceso a datos. No obstante, aquí **presentamos algunos escenarios o aplicaciones** donde resulta idóneo su uso y que te pueden servir de referencia.

- **Sistemas de notificaciones en tiempo real:** es decir, gracias a Redis, se pueden enviar actualizaciones instantáneas a los usuarios, como son las notificaciones en aplicaciones web o móviles.
- **Chat y mensajería en tiempo real:** en aplicaciones de chat, podemos utilizar Pub/Sub para transmitir mensajes a los usuarios conectados en ese momento.
- **Dashboards:** con Redis se pueden actualizar estos en tiempo real mediante datos como las métricas de rendimiento, las estadísticas de uso o el monitoreo de sistemas.
- **Colas de trabajo y procesamiento distribuido:** también se puede emplear en la distribución de tareas o mensajes a varios trabajadores o servicios que procesan datos en paralelo.
- **Juegos en línea:** Redis posibilita el manejo de comunicaciones en tiempo real entre jugadores, como los estados de juego, los movimientos y las puntuaciones alcanzadas.
- **Sistemas IoT (internet de las cosas):** aplicación para transmitir datos desde dispositivos IoT a servidores o entre dispositivos, como sensores y actuadores.
- **Integración de microservicios:** en arquitecturas de microservicios, Redis permite la comunicación desacoplada entre diferentes servicios.
- **Automatización y coordinación de procesos:** también facilita la coordinación de procesos y flujos de trabajo entre diferentes aplicaciones y sistemas.

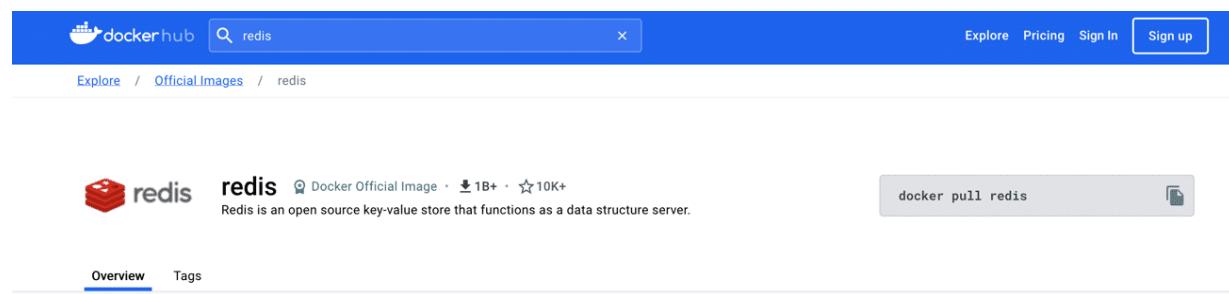
**Redis ofrece una solución robusta y de alto rendimiento para una variedad de necesidades de almacenamiento y caché en aplicaciones web. Su arquitectura simple pero potente, junto con su versatilidad en el manejo de datos, lo convierte en una buena elección para multitud de aplicaciones.**

## Instalación de Redis

Ya hemos desvelado las fortalezas de Redis, ahora toca comenzar a trabajar con él, claro que antes necesitamos preparar un entorno. Concretamente, nosotros usaremos Docker para poder trabajar con él.

**i** Como punto de referencia, contamos con la página de [documentación oficial de Redis](#), previamente a la instalación te recomiendo que la visites y te empapes de la información que comparte.

Ahora sí, ahora vamos a comenzar montando **nuestra imagen de Redis**, que la tenemos disponible en el repositorio de DockerHub.



Una vez que ya disponemos de nuestra imagen, toca **montar un contenedor y abrir el puerto** para poder conectarnos a él: `docker run -d -name redis -p 6379:6379 redis`. Con esto ya tendríamos nuestro servidor Redis básico preparado para ser usado. Sencillo, ¿verdad?

## Redis-cli: comandos básicos

Veamos ahora qué podemos hacer con Redis. Para ello, empecemos a trabajar con `redis-cli`, la línea de comando de Redis. El primer paso que debemos dar es **arrancar nuestro servidor**, ¿cierto?

```
docker exec -it 1a5cb781e753 redis-cli
```

Donde `1a5cb781e753` es el ID del contenedor donde hemos montado el servicio.



Una vez iniciado, vamos a empezar a trabajar con los pares clave:valor, el sistema base de asignación en Redis, para lo cual tenemos disponibles los siguientes comandos, ¡anota!

### SET

Se utiliza para asignar a una determinada clave, un valor específico.

```
SET mi_clave un_valor
```

**GET**

—

Cuando queremos recuperar el valor que contiene una clave, GET es nuestro comando. Si la clave no existe, devuelve 'nil'.

*GET mi\_clave***DEL**

—

Si ya no vamos a necesitar más una clave o conjunto de claves, con DEL podremos eliminarlas.

*DEL clave\_1 clave\_2 clave\_n***EXPIRE**

—

Podemos querer tener valores almacenados un determinado tiempo y que, tras pasar dicho periodo, se autoeliminen.

*EXPIRE clave num\_segundos***INCR**

—

Incrementa el valor de una clave en 1. Si la clave no existe, lo considera con valor 0.

*INCR clave*

El manejo de claves-valor ya los tenemos controlados, así que pasemos al manejo de listas, es decir, a la secuencia de cadenas que nos permiten **almacenar y manipular datos** en un determinado orden.

## **LPUSH** —

Para añadir un elemento por la izquierda.

*LPUSH lista valor*

## **RPUSH** —

Si añadimos un elemento por la derecha.

*RPUSH lista valor*

## **LSET** —

Establecemos un valor en una determinada posición (empezando por cero).

*LSET lista posición valor*

## **LPOP** —

Obtiene el primer valor por la izquierda.

*LPOP lista*

**RPOP**

—

Obtiene el primer valor por la derecha.

*RPOP lista*

**LINDEX**

—

Obtiene el valor en una determinada posición (empezando desde cero).

*LINDEX clave posición*

**DEL**

—

Al igual que pasaba con las claves-valor, DEL nos permite también eliminar listas.

Vamos un paso allá y veamos ahora los **comandos** para trabajar con los conjuntos.

**SADD**

—

Añade elementos a un conjunto.

*SADD clave elemento1 elemento2 elemento\_n*

## SISMEMBER

Comprueba si un elemento está en un conjunto (recordemos que en los conjuntos los elementos son únicos según su clave).

*SISMEMBER clave elemento*

## SREM

Elimina uno o más elementos de un conjunto.

*SREM clave elemento1 elemento2 elemento\_n*



**Recuerda:** algunas operaciones específicas del manejo de conjuntos son **unión** (SUNION), **intersección** (SINTER) y **diferencia** (SDIFF).

A continuación, vamos a **estudiar las operaciones** sobre el tipo de dato *hash*, muy útil para el manejo de objetos y elementos relacionados.

## HSET

Añade uno o más clave:valor a un hash.

*HSET clave\_hash clave1 valor2 clave2 valor2 clave\_n valor\_n*

**HGET**

Obtiene el valor asociado a un campo del hash.

*HGET clave\_hash campo*

**HGETALL**

Obtiene todos los campos y valores de un hash.

*HGETALL clave\_hash*

**HDEL**

Elimina uno o más campos de un determinado hash.

*HDEL clave\_hash campo1 campo2 campo\_n*

Y por último, conozcamos **los comandos relativos al mecanismo Pub/Sub**.

**PUBLISH**

**SUBSCRIBE**

**UNSUBSCRIBE**

Permite enviar mensajes (publicarlos) en un canal determinado.

*PUBLISH canal mensaje*

**PUBLISH****SUBSCRIBE****UNSUBSCRIBE**

Suscribe al cliente a uno o más canales.

*SUBSCRIBE canal1 canal2 canal\_n*

**PUBLISH****SUBSCRIBE****UNSUBSCRIBE**

Cancela la suscripción del cliente de uno o más canales.

*UNSUBSCRIBE canal1 canal2 canal\_n*

# Publicación y suscripción en Redis (Pub/Sub)



Qualentum Lab

---

El modelo de **publicación y suscripción** (Pub/Sub) de Redis es un patrón muy potente para la comunicación en tiempo real entre diferentes procesos, aplicaciones o componentes de una aplicación.

---

**Este modelo es especialmente útil en la construcción de sistemas de mensajería, en las notificaciones en tiempo real y en aplicaciones colaborativas.**

---

1

**Publicación (publish):** se refiere al acto de enviar un mensaje a un canal específico. Cualquier cliente conectado a Redis puede publicar en cualquier canal.

2

**Suscripción (subscribe):** los clientes pueden suscribirse a uno o más canales para escuchar los mensajes enviados a esos canales.

3

**Patrón de suscripción (pattern subscribe):** además de suscribirse a canales específicos, Redis permite suscribirse a patrones de canales.

Por ejemplo, imaginemos que a los **distintos canales de un chat** los llamamos *chat\_1*, *chat\_2*, *chat\_n*. Si queremos que un usuario se suscriba a todos los canales del chat, lo haríamos con *PSUBSCRIBE chat\_\**.

Aspectos que debemos tener en cuenta:

- Redis **no mantiene el estado** de los mensajes en los canales. Una vez que se publica un mensaje en un canal, se envía a todos los suscriptores actuales y luego se descarta.
- Cabe mencionar también que los mensajes se envían a los suscriptores **casi instantáneamente**, lo que facilita la construcción de aplicaciones en tiempo real.
- Y por último: cada canal puede tener múltiples suscriptores, y cada suscriptor puede escuchar en varios canales.

Cuando diseñamos aplicaciones con Redis basadas en Pub/Sub, tenemos que considerar distintos **aspectos**, por ejemplo:

#### ***El manejo de grandes volúmenes de mensajes***

Hay que asegurarse de que tanto el servidor de Redis como los clientes puedan manejar el volumen de mensajes esperado.

### ***El diseño de canales y patrones***

Es imprescindible planificar cuidadosamente la estructura de los canales y patrones para garantizar una distribución eficiente de los mensajes.

### ***La confiabilidad y persistencia***

Dado que Redis Pub/Sub no persiste mensajes, es imprescindible considerar mecanismos alternativos para la retención de mensajes si es necesario.

### ***La seguridad***

Como ya supones, resulta esencial implementar medidas de seguridad para proteger los canales y asegurar que solo los clientes autorizados puedan publicar o suscribirse.

# Gestión de sesiones con Redis



Qualentum Lab

La gestión de sesiones **implica mantener el estado del usuario a través de múltiples solicitudes** en un entorno sin estado como HTTP. Redis, con su rendimiento excepcional y unas estructuras de datos flexibles, se convierte en una excelente opción para la gestión de sesiones en aplicaciones web y móviles.

Además, **la capacidad de Redis** para manejar grandes volúmenes de conexiones simultáneas lo hace especialmente adecuado para aplicaciones modernas, altamente interactivas y con un gran número de usuarios.

Revisemos, a continuación, qué ventajas nos aporta en la gestión de sesiones y por qué.

## Un alto rendimiento

Al almacenar datos de sesión en la memoria, Redis ofrece tiempos de acceso muy rápidos.

## Escalabilidad

Redis escala bien horizontal y verticalmente, por lo tanto, resulta idóneo para aplicaciones con un gran número de usuarios.

## Persistencia configurable

Aunque Redis es una base de datos en memoria, ofrece opciones de persistencia para garantizar que los datos de sesión no se pierdan, en esos casos en los que resulte necesario conservarlos.

## Estructuras de datos flexibles

Como ya sabemos, nos permite almacenar sesiones como strings, hashes, listas, etc., por lo tanto, también nos ofrece una gran flexibilidad en cómo se estructuran los datos de sesión.

# Implementación de la gestión de sesiones

Para estudiar de manera clara y concisa la fase de implementación, la dividiremos en cuatro procesos que pintaremos con rápidas pinceladas.

1

## Almacenamiento de datos de sesión

- Cada sesión se puede representar como un hash en Redis, con la ID de sesión como clave.
- Se pueden almacenar diversos datos, como identificadores de usuario, preferencias, y estados de autenticación.

2

### Creación y recuperación de sesiones

- Al iniciar una sesión, se genera una ID única que se utiliza como clave en Redis.
- Los datos de sesión se almacenan y recuperan utilizando esta ID.

3

### Expiración y limpieza de sesiones

- Redis permite establecer un tiempo de vida (TTL) para las sesiones, después del cual la sesión expira automáticamente.
- Esto ayuda a liberar memoria y mantener limpio el almacén de sesiones.

4

### Integración con aplicaciones web y móviles

- Muchos frameworks web y móviles ofrecen integración directa con Redis para la gestión de sesiones.
- Esto permite una implementación y gestión de sesiones eficiente y segura.

## Consideraciones de seguridad y mejores prácticas

Para finalizar el apartado dedicado a las publicaciones y suscripciones con Redis, comparto aquí una serie de consejos.



**Protege los datos de sesión:** es necesarios asegurarse de que los datos sensibles almacenados en las sesiones estén protegidos, posiblemente mediante cifrado.



**Acceso seguro a Redis:** debemos configurar el servidor Redis para que solo las aplicaciones autorizadas puedan acceder a los datos de sesión.



**Gestión de IDs de sesión:** hay que evitar la generación predecible de IDs de sesión para prevenir ataques, por ejemplo, la fijación de sesión.

En cuanto a las buenas prácticas, destacamos estas tres:

1

**El monitoreo del uso de la memoria.** Dado que las sesiones se almacenan en memoria, es importante monitorear el uso de la memoria de Redis y escalar según sea necesario.

2

**Manejar sesiones inactivas.** Es fundamental implementar políticas para manejar sesiones inactivas o abandonadas, así evitaremos el consumo innecesario de recursos.

3

Y por último, **la sincronización con la base de datos principal.** En algunos casos, puede resultar necesario sincronizar los datos de sesión con la base de datos principal para garantizar la coherencia de estos.

# Interacciones de aplicaciones y caché



Vale, ya sabemos qué es un sistema de caché, más concretamente Redis, pero ¿y si un dato cambia?, ¿y si no tenemos un dato en caché? ¿y si el servidor se cae?

Vamos a ver algunos escenarios dentro de los casos más típicos y cómo interactúan las aplicaciones con Redis.

## **Datos no presentes en caché**

Si la aplicación solicita un dato que no está en el caché de Redis, entonces la aplicación busca el dato en la fuente de datos principal (como una base de datos). Una vez recuperado el dato, la aplicación lo almacena en Redis para futuras solicitudes, mejorando así el tiempo de respuesta para ese dato.

## **Datos presentes en caché**

Imaginemos ahora que la aplicación solicita un dato que ya está almacenado en Redis. ¿Qué hace este?

Redis devuelve inmediatamente el dato, evitando la necesidad de acceder a la base de datos principal; por lo tanto, se reduce la carga en la base de datos y acelera el tiempo de respuesta.

## Actualización de datos y reflejo en caché

Supongamos ahora que la aplicación actualiza un dato en la base de datos principal. Aquí podemos optar por varias opciones.

- **Borrado:** la aplicación elimina el dato correspondiente de Redis. En la próxima solicitud, como el dato ya no está en caché, se sigue el proceso de datos no presentes en caché que vimos antes.
- **Actualización:** la aplicación actualiza el dato tanto en la base de datos como en Redis simultáneamente.

## Expiración de datos en caché

Si un dato en Redis se establece para expirar después de un tiempo determinado, una vez expirado este, Redis lo elimina automáticamente. ¿Cuál es la consecuencia? En la próxima solicitud, la aplicación experimenta un dato no presente en caché y recupera y almacena nuevamente el dato en Redis.

 **Recuerda:** la elección de cómo interactuar con Redis dependerá de los requisitos específicos de la aplicación y del problema que se esté tratando de resolver.

## Otros casos de uso

A continuación, mediante un **resumen esquemático**, vamos a explorar algunos casos de uso de Redis según distintos patrones de diseño de aplicaciones web.

### Caché de resultados de consultas

- **Contexto:** imaginemos que almacenamos los resultados de consultas de bases de datos en Redis para acelerar el acceso a datos que se consultan frecuentemente.
- **Implementación:** utilizaremos el comando SET para almacenar el resultado de una consulta y GET para recuperarlo. Se puede usar una política de expiración para asegurar que los datos no se vuelvan obsoletos.
- **Casos de uso:** muy útil en aplicaciones donde ciertas consultas a la base de datos son costosas y los datos no cambian con frecuencia.

### Caché de sesiones

- **Contexto:** vamos a utilizar Redis para almacenar datos de sesión, proporcionando acceso rápido y reduciendo la carga en el almacenamiento principal.
- **Implementación:** almacenamos la información de la sesión como un hash en Redis, utilizando el ID de sesión como clave.
- **Casos de uso:** ideal para aplicaciones web con un alto volumen de usuarios, donde la gestión rápida y eficiente de las sesiones es crítica.

## Colas de mensajes

- **Contexto:** queremos crear colas para gestionar tareas en segundo plano o para sistemas de mensajería.
- **Implementación:** usamos las listas de Redis para crear colas, insertando elementos con LPUSH y procesándolos con RPOP.
- **Casos de uso:** idóneo para aplicaciones que necesitan procesar tareas en segundo plano, como el envío de correos electrónicos, la generación de informes, etc.

## Contador de tasa de límite (rate limiting)

- **Contexto:** queremos limitar la frecuencia de ciertas operaciones, como las solicitudes a una API.
- **Implementación:** utilizamos los comandos como INCR para incrementar contadores y establecer un límite en el número de operaciones permitidas en un intervalo de tiempo.
- **Casos de uso:** el contador de tasa de límite resulta muy útil en APIs públicas para prevenir el abuso y garantizar la equidad en el uso de recursos.

## Almacenamiento de datos en tiempo real

- **Contexto:** necesitamos almacenar y recuperar datos en tiempo real para aplicaciones como chats o juegos en línea.
- **Implementación:** utilizaremos estructuras como las listas, los conjuntos o los canales de publicación y suscripción para manejar datos en tiempo real.
- **Casos de uso:** idóneo para chats, tableros de anuncios en vivo, juegos en línea..., donde la actualización en tiempo real es fundamental.

## Publicación y suscripción (Pub/Sub)

- **Contexto:** necesitamos implementar un sistema de comunicación donde los productores publican mensajes y los consumidores se suscriben a canales para recibir esos mensajes.
- **Implementación:** utilizaremos el comando PUBLISH para enviar mensajes y el comando SUBSCRIBE para escuchar en un canal.
- **Casos de uso:** esta implementación es idónea para sistemas de notificación y la mensajería en tiempo real, donde se requiere una comunicación eficiente entre diferentes partes de una aplicación.

## Gestión de configuraciones

- **Contexto:** se requiere almacenar las configuraciones de la aplicación que pueden cambiar dinámicamente.
- **Implementación:** almacenamos las configuraciones como claves en Redis, permitiendo que la aplicación las lea y actualice en tiempo real.
- **Casos de uso:** esta implementación funciona con aplicaciones que necesitan una alta flexibilidad en la gestión de configuraciones, donde los cambios deben reflejarse instantáneamente sin reiniciar la aplicación.

# Conclusiones



Qualentum Lab

A modo de resumen y para destacar los contenidos más relevantes que hemos compartido en este fastbook, quédate con los siguientes aprendizajes.

## Recuerda cuándo es aconsejable usar Redis

- Con datos de acceso frecuente, cuando se requiere acceso rápido a datos que se consultan con frecuencia.
- Con estructuras de datos complejas, para aprovechar sus estructuras de datos avanzadas en operaciones que van más allá de simples claves-valor.
- En contextos que requieren escalabilidad y alta disponibilidad.
- Con aplicaciones en tiempo real, donde la velocidad es crítica, por ejemplo, en chats, juegos en línea o dashboards con dicha necesidad de inmediatez.

## Recuerda también cuando no es aconsejable usar Redis

- Cuando hay un gran volumen de datos persistentes, es decir, si los datos son demasiado grandes para caber eficientemente en la memoria.
- Si la consistencia de datos es crítica o, dicho de otra manera, en casos donde se requiere una consistencia absoluta de datos y tolerancia cero a la pérdida de datos.
- En búsquedas complejas de datos. Lamentablemente, Redis no es ideal para escenarios que requieren consultas complejas, como búsquedas basadas en múltiples atributos.

Como hemos visto, Redis es **una herramienta muy potente y flexible** para el desarrollo de aplicaciones web modernas, sobre todo en cuanto a mejorar el rendimiento, manejar los datos en tiempo real y buscar la escalabilidad. Sin embargo, es importante evaluar las necesidades específicas de cada proyecto para determinar si su implementación es idónea o no.

¡Enhорabuena! Fastbook superado



[Qualentum.com](http://Qualentum.com)