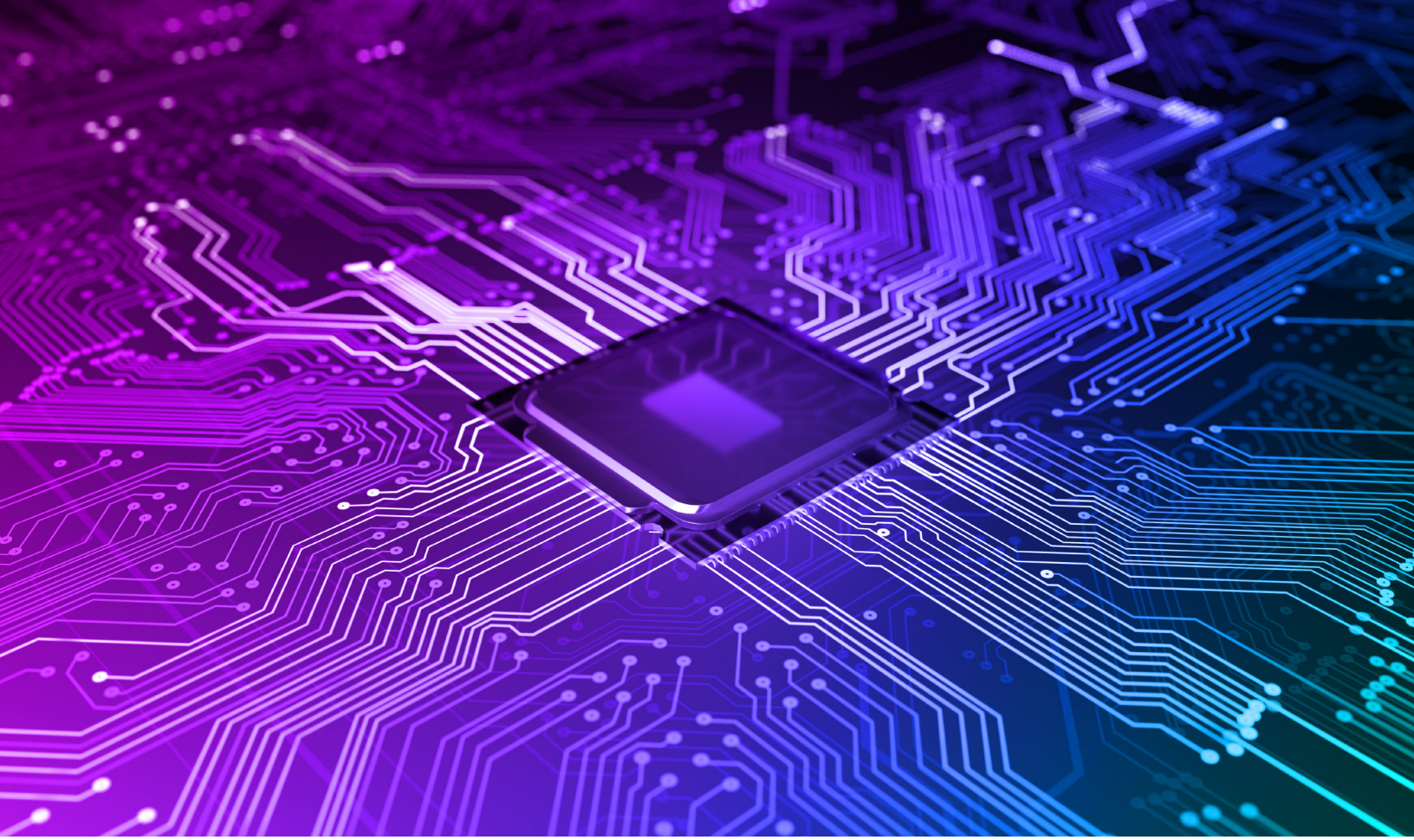


Solution Brief

Redis Enterprise for Caching

Enable scalable, resilient, real-time applications

redis

Digital is the new storefront for sales, an instrument for innovation, and the frontline for customer engagement. Customers now primarily interact with, or experience business services, through applications. And they expect immediate responses. Real-time applications start with fast data performance.

And that's where Redis Enterprise can make a huge difference.

THE CHALLENGE

Achieving speed and scalability with current databases

It isn't easy to achieve the data performance needed to match user expectations. Businesses cope with legacy infrastructure, slow databases, and technical debt. They regularly face cost, effort, and business continuity constraints.

Data is the lifeblood of business applications. Yet the organizations' applications have to process large datasets, which slows down performance – particularly when the application has to access a database every time it's needed.

The outcomes are painful and expensive:

- **Latency:** Customers expect applications to respond Instantly, but many business applications are simply too slow.
- **Scalability:** Many applications and databases struggle to support the growth of digital business services or to stand up to sudden demand spikes.
- **Resilience:** Because apps are at the forefront of business downtime is a significant business risk—and preventing downtime is a major technology challenge.
- **Technical debt:** Businesses need to add performance, scalability, and resilience without enduring the pain and expense of redesigning a system from scratch.

THE SOLUTION

Caching with Redis Enterprise

An enterprise-grade caching layer makes a huge difference.

A caching layer is a temporary storage location that sits between an application and the database. It allows an application to quickly access frequently needed data without having to query the database every time. This significantly enhances application performance and scalability by improving response times and decreasing the burden on databases.

This isn't merely a technical matter. Caching offers a business advantage, because it eases the demands on the software development and operations staff. Implementing enterprise-class caching avoids time-consuming, resource-intensive, and expensive database migrations or application refactoring. In other words: This is a fast way to solve performance and availability problems without asking the development staff to add yet another project to their To-Do lists.

Built by the creators of Redis Open Source, Redis Enterprise has over a decade of deployment in challenging enterprise production environments.

Available on-premises, with Kubernetes, in hybrid environments, or fully managed in Amazon Web Services (AWS, Microsoft Azure, and Google Cloud (GCP), Redis Enterprise helps thousands of huge organizations maintain real-time performance with scalability, cost efficiency, and high availability. It's easy to work with, too, which is why developers clamor to use it.

Common Redis Enterprise caching patterns

The appropriate caching pattern to deploy depends on business and application needs. Redis Enterprise supports whichever pattern makes sense for you: cache-aside, query caching, write-behind, write-through, and cache-prefetching.

Which should you use? The criteria include such matters as whether an entire dataset or a subset needs to be cached; if the workload is read-heavy or write-heavy; and the importance of data consistency.

When to use common Redis Enterprise caching patterns:

- **Cache-aside:** to speed up reads typically used with least-recently-used (LRU) eviction to reduce the costs of caching all the data when cache-misses are acceptable.

  Real world example: A featured product on a retail site. The first time the product is featured, details (image, price, and metadata) are loaded from the system of record and the application would then cache details into Redis Enterprise to serve the same data to a high volume of users with low latency.

- **Query caching:** when there is a need to speed up high-latency commonly used queries with minimal overhead

  Real world example: When speeding up often-repeated SQL queries or when upgrading legacy applications to microservices environments without replatforming databases

- **Write-behind caching:** to speed up write-heavy workloads without forcing the application to manage multiple data source connections

  Real world example: Processing a financial transaction in Redis Enterprise and keeping financial records and transactional data in cold storage for auditing or reporting purposes

- **Write-through caching:** to speed up reads when consistency without forcing the application to manage multiple data source connections

  Real world example: When businesses need to cache but absolutely cannot lose data, especially common with critical retail customer data or financial transaction data

- **Cache prefetching:** for continuous replication between write-optimized and read-optimized workloads by offloading reads from the system of record

  Real world example: Cache-prefetching user profiles, allowing frequent writes to occur directly in the database, while preventing preloading data into Redis Entreprise to support a high volume of reads

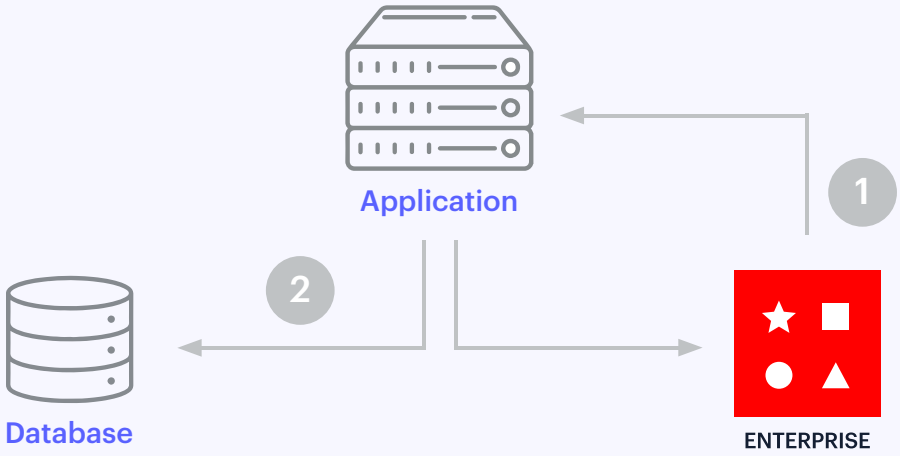
Cache-aside

When to use it: To speed up database reads. Cache-aside is typically used with least-recently-used (LRU) eviction.

Real world example: Consider the presentation of a featured product on a retail website. The first time the product is featured, the e-commerce application loads the product details (image, price, and metadata) from the system of record. The application then caches details into Redis Enterprise. The result: the website can serve the same data to a high volume of users with low latency.

Description: The most common way to use Redis Enterprise as a cache is cache-aside. Data is loaded into the Redis Enterprise cache only when necessary – hence it sometimes is called lazy loading.

Cache-aside is a common choice for read-heavy applications when a subset of a dataset needs to be cached, and when cache misses are acceptable. With a cache-aside pattern, the application handles all data operations, and it directly communicates with both the cache and database. The database and cache do not communicate directly with each other.



How cache-aside works with Redis Enterprise

1. The application looks into the Redis Enterprise cache to retrieve data
   - If the data is found (called a cache hit), Redis Enterprise delivers the data to the application. This happens with sub-millisecond latency
   - If the data is not found (a cache miss), the application retrieves the data from the database. This happens with higher latency.
2. The application then writes data to Redis Enterprise so it is available in the cache for future retrieval.

Query caching

When to use it: To speed up high-latency commonly-used queries

Real world example: Bringing legacy applications up-to-date is a painful project to contemplate; nobody wants to touch dusty, finicky code even when older systems have to interact with new software. Query caching lets you leave the legacy system in place, and connect it to newer systems with faster SQL data performance.

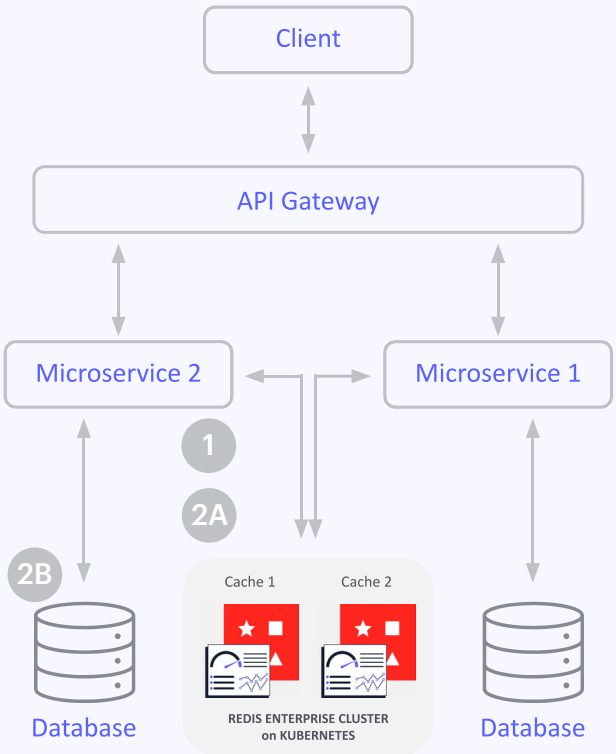
Description: Use query caching to improve query performance in order to improve response times for data queries.

Query-caching is a unique implementation of the cache-aside pattern, devise to speed up (often repeatable) SQL queries against a slower system of record. With query caching, you don't need to transform data into another data structure. From the application's point of view, it's just a database query, where the value returned comes from the cache instead of a deliberate database read. The query is used as the key and the serialized result set as the value.

One common use is among businesses that are migrating to microservices. Using query caching is common when businesses migrate to microservices without replatforming current systems or starting over from scratch. One element in such a solution is Redis Smart Cache. It enables developers to quickly deploy a standardized query cache to simplify management and operations – and you don't need to redesign an application. Just add the library as a new dependency, with your Redis endpoint as its property file. The library also enables analytics on all queries flowing through a Java Database Connectivity (JDBC) driver to the system of record.

How query-caching works with Redis Enterprise

1. When an application wants to query a database, it sends a SQL query to a Redis Enterprise cache.
2.
   A. If the data is found (called a cache hit), Redis Enterprise delivers the data to the application. This happens with sub-millisecond latency
   B. If the data is not found (a cache miss), the application retrieves the data from the database. This happens with higher latency.



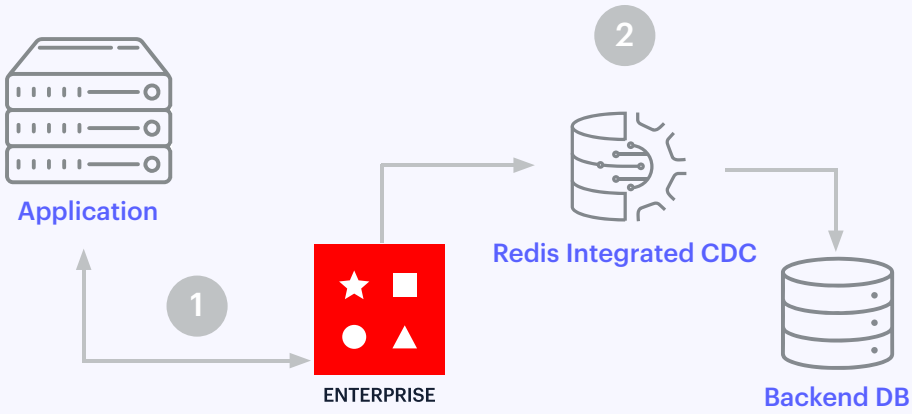
Write-behind caching

When to use it: To speed up reads when consistency is critical, without forcing the application to manage multiple data source connections

Real world example: Some industries require databases for additional record-keeping, such as medical and financial firms. But writing to multiple databases per transaction inhibits write response times and can lead to application latency. Write-behind caching is a good option for, say, when you need to process and keep financial records and transactional data in cold storage for auditing or reporting purposes.

Description: Some applications access existing information far more often than they store or update transactions. Cache-aside and query caching are good for those scenarios. However, when an application performs a lot of writes to the database and those transactions need to happen quickly, write-behind caching is a better choice. It improves write performance and eases application development since the application writes to only one place: Redis Enterprise. Redis Enterprise then asynchronously updates the backend database.

You can use Redis Enterprise's integrated Change Data Capture (CDC) alongside the cache to identify changed data in the cache, and ensure that data is eventually updated in the underlying database.



How write-behind caching works with Redis Enterprise

1. The application writes data to the Redis Enterprise cache, not directly to the database.
2. After the data is stored in Redis Enterprise, Redis Enterprise asynchronously writes to the database using integrated CDC.

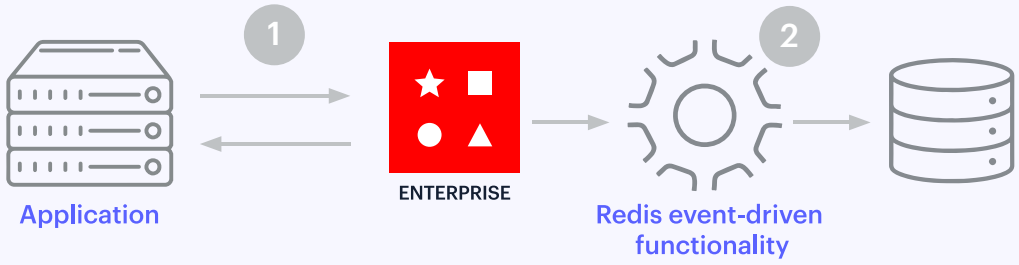
Write-through caching

When to use it: To speed up write-heavy workloads without forcing the application to manage multiple data source connections.

Real world example: Some businesses use "mission critical" as a buzzword. But for many, it's a stark statement of fact – and compromises are not an option. Consider write-through caching when you need to cache but absolutely cannot lose data, such as with critical retail customer data or financial transaction data.

Description: Write-through cache strategy is similar to the write-behind approach, as the cache sits between the application and the operational data store. The difference is that, with write-through caching, the updates are done synchronously. An application updates the cache, and the cache takes care of updating the database immediately. The application can then read data directly from Redis Enterprise with latency measured in sub-milliseconds.

The write-through pattern favors data consistency between the cache and the data store. It relies on event-driven functionality to implement data flows between systems. Redis has the ability to ensure that a database is updated synchronously to maintain consistency between the cache and system of record.



How write-through caching works with Redis Enterprise

1. The application writes data to the Redis Enterprise cache.
2. Redis Enterprise ensures that the system of record is updated synchronously to maintain consistency between Redis Enterprise and system of record.

Cache prefetching

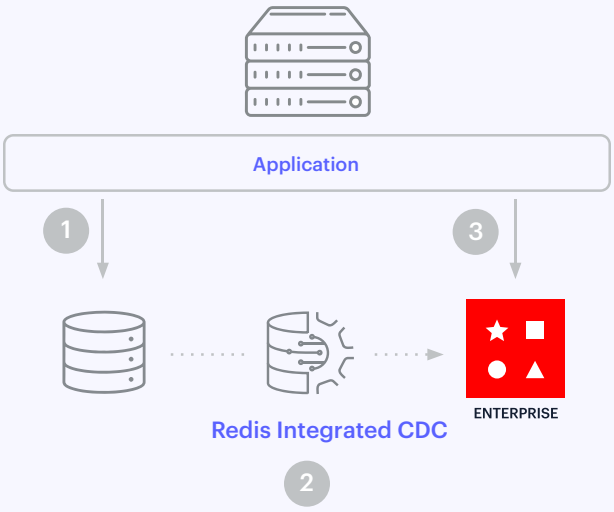
When to use it: To deploy continuous replication between write-optimized and read-optimized workloads by offloading reads from the system of record.

Real world example: Applications rely on user profiles to identify individuals throughout the duration of their sessions, beginning with authentication. These user profiles are read from frequently but rarely change. Cache-prefetching user profiles for applications, allows infrequent writes to occur directly in the database, while preloading data into Redis Enterprise to support a high volume of reads. This leads to more responsive applications, increased scalability, and lower costs.

Description: Cache prefetching is used for continuous replication between write-optimized and read-optimized workloads. With this caching pattern, writes from the application occur directly to the database. Data is replicated to Redis Enterprise as it changes in the system of record, using integrated CDC, so it arrives in the cache before it needs to be read by the application.

How cache prefetching works with Redis Enterprise

1. The application writes data directly to its original system of record.
2. Integrated CDC replicates data to the Redis Enterprise cache as it changes in the system of record. The net effect is that the data arrives before the application asks for it in order to perform a database read.
3. The application then reads directly from Redis Enterprise.



Cache with confidence

Many software projects start as grassroots applications. With success, they grow into extensive systems on which thousands or millions rely. As the number of users, volume of data, and geographic locations increase, so do issues with cost, scalability, operations, and system availability.

Most caches lack the functionality to alleviate these issues. Developers spend time spinning their wheels or reinventing those wheels, which is not a good use of their time – particularly when that energy is better spent on improving the innovative software. The cache provided by Redis Open Source is great, and we're proud of it – but at some point, it makes sense to invest in tools built by those with subject matter expertise. (That would be us.)

Failing to upgrade to a better caching solution can hamper business growth, lead to data loss or poor application performance, and incur the opportunity costs of significant effort and expense going toward Redis operations. Upgrading to an enterprise-grade cache is critical to enabling and supporting business growth.

We built essential caching functionality – (so you don't have to)

While caching is a simple enough concept, in practice it can be very complicated. Businesses need to manage time to live (TTL), data consistency, and scaling, as well as many other variables across large, complex data ecosystems.

Redis Enterprise is the only enterprise-grade cache that provides:

- **Flexibility:** A single data platform that seamlessly works alongside existing infrastructure, on premises, in hybrid environments, or in any cloud
- **Scalability:** Effortless scaling that maintains sub-millisecond latency at millions of operations per second
- **Resilience:** Battle-tested availability with a five-nines SLA that excels in supporting mission-critical applications
- **Cost efficiency:** Storage tiering can save up to 80% on infrastructure costs when caching large datasets
- **Simplicity:** Works alongside existing architecture to support a variety of caching patterns—without re-architecting or replatforming—and a well-earned reputation for being blissfully easy to work with

Redis has solved just about every type of caching problem out there. As we developed this expertise, we built functionality that makes caching easier for our customers. With Redis Enterprise, we handle caching's hidden complexity so you don't have to deal with it yourself.

You have other things to do. Let us take care of this.

When you cache with Redis Enterprise, you can:

- **Mirror data between your cache and system of record:** With **integrated CDC**, you can identify and deliver changes in real time to Redis Enterprise from asystem of record
- **Simplify caching in the most complex environments:** Use **Redis Smart Cache** to quickly and seamlessly deploy and standardize numerous query caches in complex microservices environments—without the need to change application code to support a variety of caching patterns—without re-architecting or replatforming—and a well-earned reputation for being blissfully easy to work with
- **React to events in Redis Enterprise:** Use event-driven functionalit for write-through caching and improve application performance while guaranteeing that all changes are written to your backend databases

Learn more

Dive deeper into caching. Explore common caching patterns, how they work, when to use them, and how they can benefit you. Read the definitive guide to Caching at Scale with Redis.