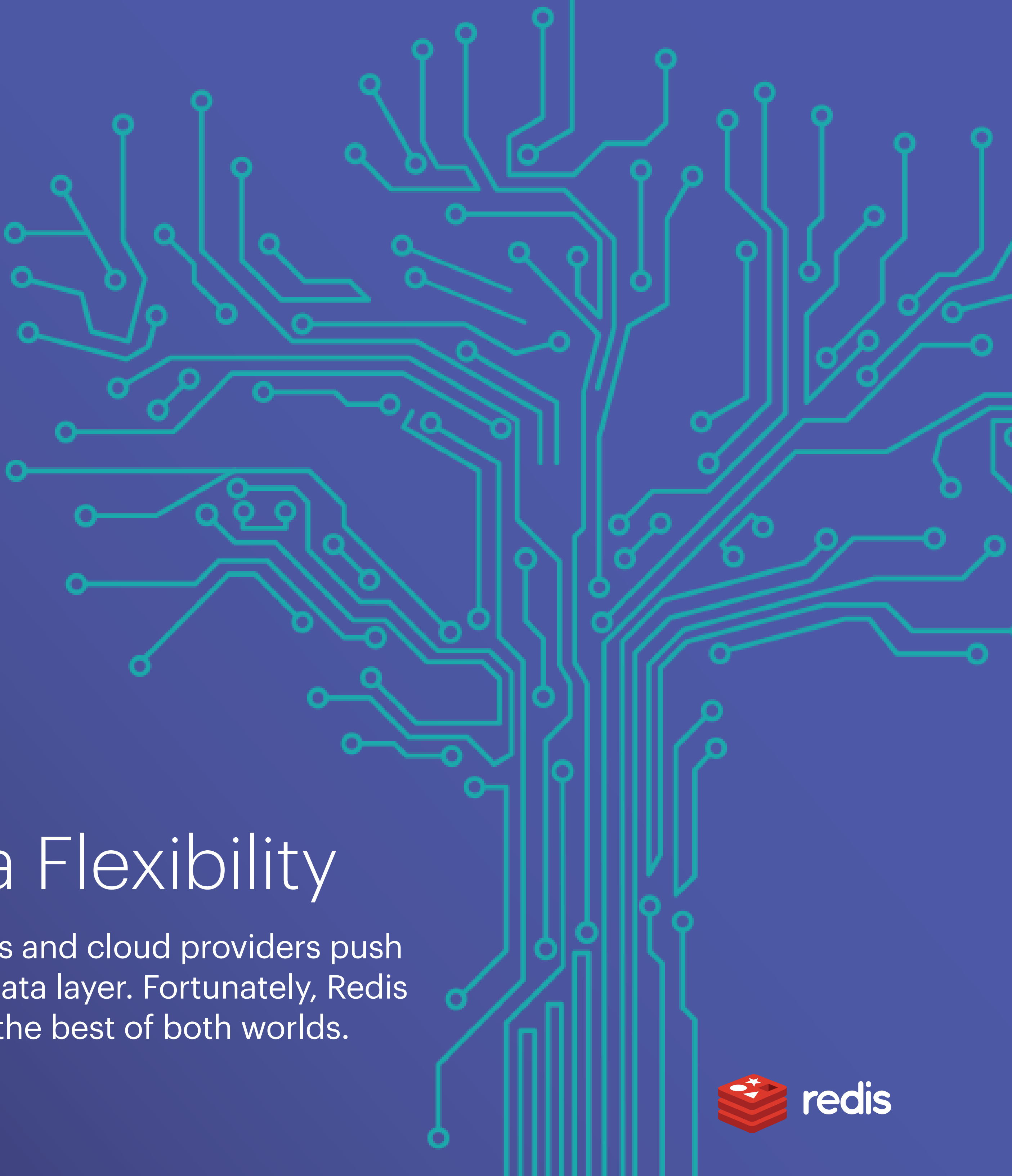
Strategic Data Flexibility

Both old-line database vendors and cloud providers push compromised models of the data layer. Fortunately, Redis Enterprise offers a third way—the best of both worlds.



redis

Executive summary

Picking the right database is a critical decision, with a cascade of consequences felt across the organization and up and down the tech stack for years and even decades. Unfortunately, database choice can be difficult and complex, with the full impact becoming clear only when the database is pushed at scale or beyond the initial data model.

Ironically, a limited set of options makes database choice even more complex. The database industry is largely divided into two camps. One camp advocates for a monolithic relational database model, while the other pushes customers to use a variety of purpose-built databases for every application. Think of the first camp as a hot-dog cart: they offer one thing, hot dogs. The second camp is like a food court, you select from an array of options that may or may not go together.

Relational databases = hot dogs

The hot dog vendors rely on a single data model (relational) and a single kind of software (DBMS). No matter the actual type of data being stored and queried, it has to be bent and formed to fit the relational model. The development team must write this extra layer of software to adapt a variety of data problems into relational/tabular problems. Operationally, this often produces performance or scale problems as it forces the database to do things it wasn't truly designed to handle. No matter how many condiments they ladle on top, these vendors are offering only hot dogs.

"This cart sells hot dogs. Even if you add ketchup, mustard or even pickles, it's still a hot dog. Don't let anyone tell you otherwise."

“Redis Enterprise offers a third way that delivers the best of both worlds.”

Cloud databases = food courts

The food court camp view is that many data models are required, even in a single application, which means everyone needs to use multiple, purpose-built databases. These databases are strung together across a network, yielding a fragile, slow, rat’s nest of interconnected services. Developers must learn a bewildering array of query languages, connection methods, and client libraries. And they have to worry about synchronization issues and failure tolerances when managing data across multiple services. Operations teams have to deal with configuring, scaling, securing, and maintaining not just one database platform, but several.

Both choices are compromised. They embed conflict and brittleness as well as performance and maintainability concerns. Even more importantly, it’s a false, forced choice. There is another path, one that can help you avoid the rat’s nest and having to fit everything into a single data model.

Redis Enterprise = a third way

Redis Enterprise presents a third way—a single operational interface in which you can deploy multiple databases. The databases in Redis Enterprise offer modules that extend the key-based functionality of Redis to more-specific data models such as graph, search, document, and time series, as well as probabilistic, event-triggered, and AI-serving capabilities. Developers interface with the database through standard Redis client libraries. Instead of a hot dog cart or food court, Redis Enterprise offers a full menu of well-crafted, well-considered options from a single restaurant.

Introduction

At the heart of most modern software is data. The database is one of the key technology choices for technology leaders. This choice has been informed by the development of storage itself. The earliest data storage was physical, literally holes in cards or paper tape. Over the years, we moved into magnetic media and eventually into solid-state storage devices. Today, the sophistication is evolving from the physics of how bits are stored on physical media to how the data is modeled and translated into usable information.

Yet much old-line database software was designed in an era where the levels of abstraction available today just didn't exist—software could be delivered only on physical media.

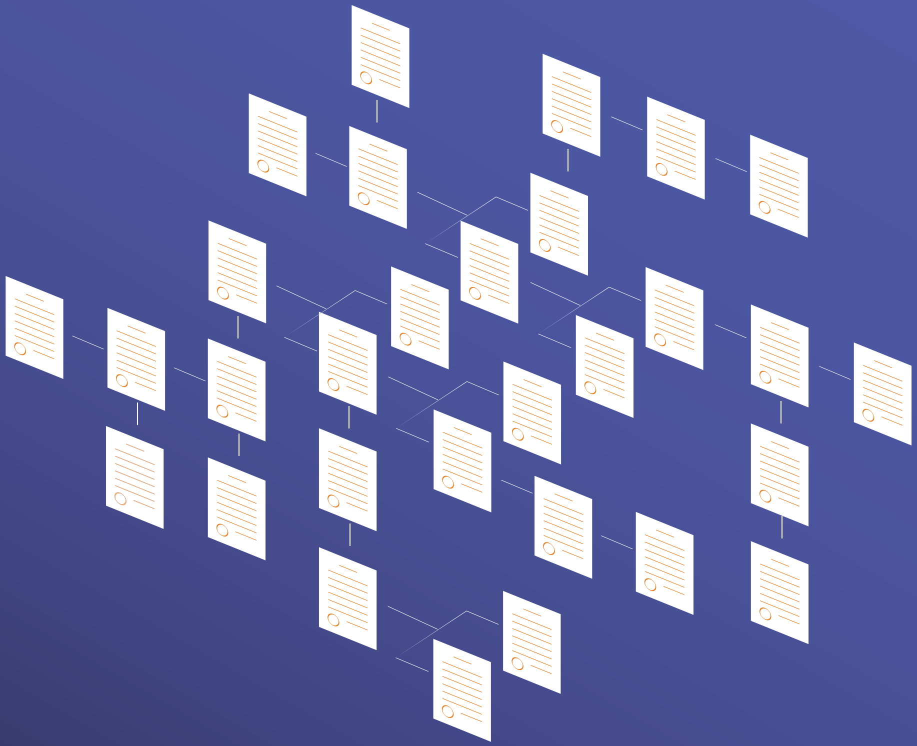
There is still a massive installed base of old-line databases that even non-legacy systems. These databases rely upon a single, monolithic, relational model. While that model remains useful for modeling some data problems, it represents a (now) genericized storage engine that has too often been used and misused in the place of proper data modeling techniques.

Cloud vendors, meanwhile, are abstracting away the notion of machines entirely, delivering purpose-built databases solely available on their service. You "just" connect them to your application. Need a database with a different model? "Just" spin up a new database and connect to it.

Critically, both the cloud and legacy databases include fundamental compromises. In many cases, they result in cyclical adoption of different technology approaches in repeated attempts to ease the pain of one set of constituents that end up coming at the expense of other constituents.

To be clear: compromises in your database strategy can lead to internal problems (staff satisfaction and accomplishment, project longevity) and external problems (performance and stability). Thankfully, the third option offers a middle way that avoids the pitfalls of legacy databases and the cloud vendors. A strategic option flexible enough to adapt to new data models natively and to allow for operational simplicity and stability.

"Much old-line database software was designed in an era where the levels of abstraction available today just didn't exist."



"Countless watts of energy were wasted as massive relational databases were launched and maintained only to never do any operation more complex than look up items by a primary key in a single table."

Database industry positions

Given the two competing approaches to database design, it's not surprising that much of the database industry falls into two camps: old-line database providers and cloud vendors. Each camp sees the landscape very differently and advocates for very different solutions.

Old-line database providers

Early on in computing, people started thinking about how to store data in a standardized format. Charles W. Bachman had created the Integrated Data Store at General Electric by 1963, representing the first thing we might recognize today as a database. A few years later, in 1970, Edgar Codd wrote "A Relational Model of Data for Large Shared Data Banks," which functioned as the blueprint for the relational database model. Larry Ellison took the ideas from this paper as the foundation for what would become the Oracle database, released in 1979.

The relational model proved to be very flexible (for the time) and eventually became the de facto database standard. The relational model and database became functionally equivalent in most conversations. Indeed, this powerful model lets you accomplish many different things with a single interface. No matter if you had a simple project for a few users or a complex organization-wide data project, the relational model was the answer. Of course, countless watts of energy were wasted as massive relational databases were launched and maintained only to never do any operation more complex than look up items by a primary key in a single table. The intention wasn't to be wasteful, but for decades, relational databases were the only viable, well-known, and well-understood option, so they got dragged into service for just about everything.

Data modeled in completely different ways was jammed into the relational database as well, creating modeling headaches. Mapping graph data onto a relational database, for example, creates indexing issues and massive performance problems. Likewise, time-series data is a write-optimized, aggregation-heavy workload that many relational databases simply could not handle in real-time.

Document storage in a relational database attempts to shove these kinds of free-form, schema-free data into a highly typed and structured tabular format. All of these data models are completely legitimate, but the ubiquity of relational databases tended to push the complexity and model adaption up into the application. Often, the result was a relational database containing data that could be accessed only by a specific application, and an overburdened database fighting to keep up with a usage pattern it was never designed to handle.

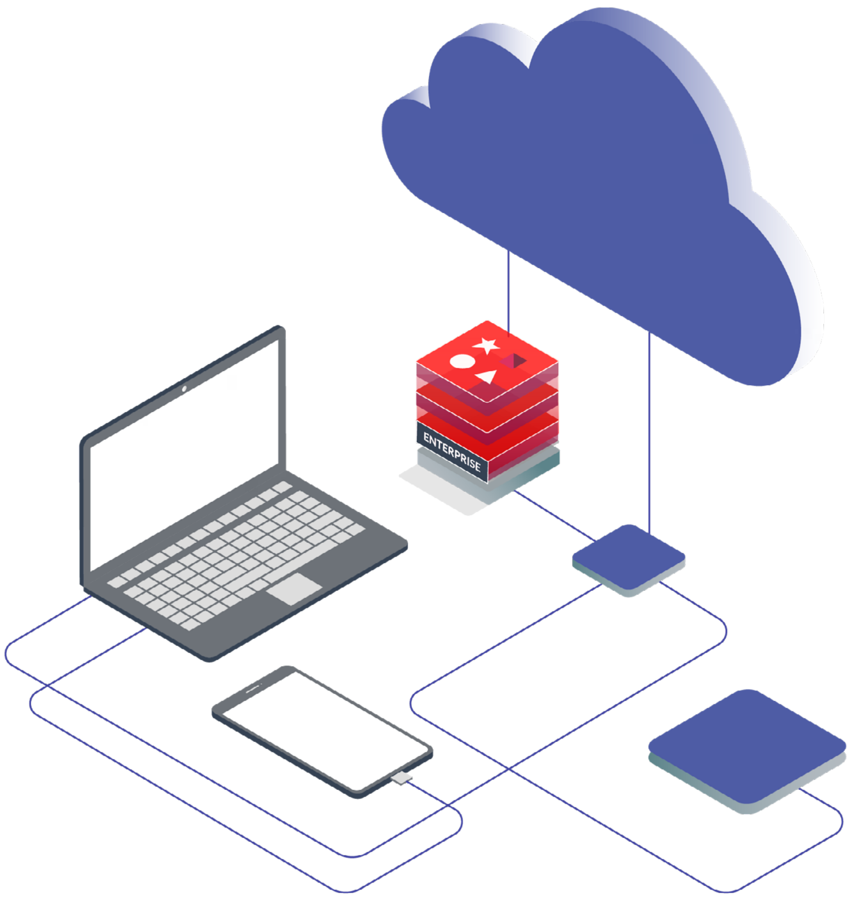
Again, this is not to say that the relational model is completely inappropriate. Like any data model, there are problems for which relational databases are the optimal choice. Applications that have highly relational data and/or cannot be effectively denormalized are still great fits for these kinds of databases. But just as when all you have is a hammer everything looks like a nail, if all you have is a relational database all problems appear relational.

To be fair, settling on a single database platform does allow for some operational benefits. The professionals running these platforms learned quickly and deeply how to keep them stable. As large, complex pieces of software, relational database systems could be run on an organizational level rather than be deployed specifically for an individual piece of software, centralizing complexity and expertise. While the one-size-fits-all approach has the potential to cause development problems, it also created a single, known vector to operationalize.

Architecturally, though, the roots for these systems date from an era prior to the birth of many people who are operating them—before the modern internet, much less cloud computing. Granted, there is little, if any, source code that originates from the late 1970s still running in this class of software, but the underlying concepts were born in a fundamentally different age.

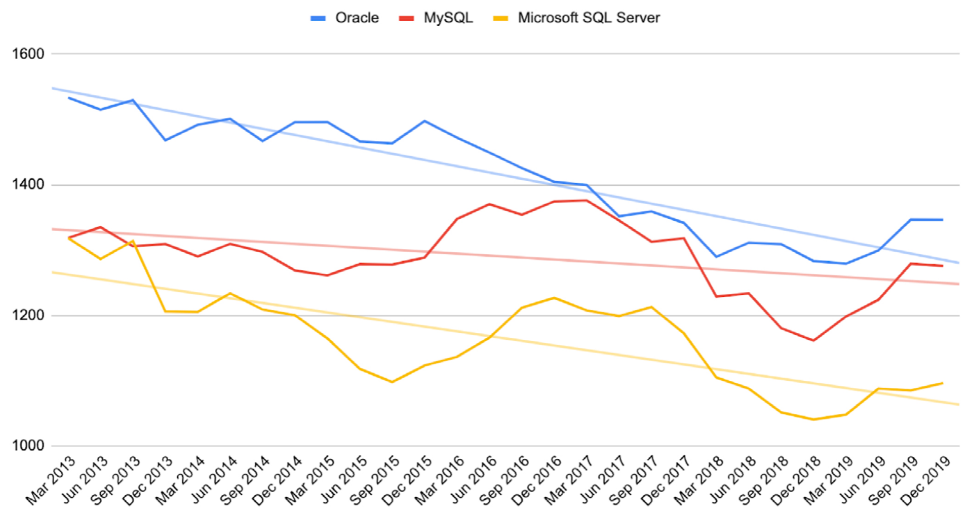
Enter the cloud

When companies started to move software to the cloud, many tried to do so without rearchitecting their applications (known as “lift-and-shift”). Unfortunately, assumptions made at the architectural level for many pieces of software did not match the drastically different cloud environment—and RDBMSes were no exception. Decades of low-level and micro optimizations had produced software that ran with acceptable performance in on-premises environments. But when placed into a virtualized environment, new challenges emerged; lower performance off server network attached storage, ephemeral compute resources, and lack of automation created a rocky path to the cloud. In short, the early lift-and-shift exercises proved that operational professionals not only need to be experts on massive database software but also on the cloud they’re using. The world had changed.



"Old-line database providers sought to provide a database to an organization, while cloud vendors seek to supply many databases to a single application."

Top 3 Relational Databases DB-Engines Scores, Quarterly 2013-2019



The modern era of cloud computing has affected the popularity of old-line databases. While relational databases hold the top three positions in DB-Engines.com rankings, their trend over time is clearly downward. Since DB-Engines started tracking the scores in 2013, there has been an overall decline in the popularity of the top relational models, according to the DB-Engines scoring methodology.

Cloud shadows: Relational databases remain popular... for now

Cloud vendors

On the other end of the spectrum sit the cloud vendors. Based on their product lines, the cloud vendors seem to reject the idea that the relational model is the standard way to store and retrieve data. The various data models represented in the cloud vendors' database products suggest that your application's data should not bend for the available database, but instead you should select the right database(s) for your application.

The cloud vendors can be seen as supporting NoSQL, a term coined as early as 1998, but with a contemporary meaning dating from 2009. NoSQL databases originally focused on providing a single data model to support a shorter list of use cases. They are purpose built for a single data model. However, any application of sufficient complexity probably has a wide range of data that should be modeled in different ways, so a single application can require multiple different databases. Indeed, where old-line database providers sought to provide a database to an organization, cloud vendors may seek to supply many databases to a single application.

Multiple databases in a single project can help solve developers' problems. Relational databases create results that require developers to rely on libraries or their own code to marshal and unmarshal data to fit the relational model (the object-relational impedance mismatch problem). The inputs and outputs of NoSQL databases tend to resemble how the application code models data internally. So, as a developer working with purpose-built databases, the proportion of time allotted to writing relevant, problem-solving code is greater compared to relational databases where there is a hidden requirement to massage the data into a usable form.

But using multiple purpose-built databases in a single application comes with its own downsides. Like any tool, there is skill and techniques of use involved; each database requires developers to learn not only the conceptual basis of the database and model, but also the accompanying libraries. For example, understanding the connection parameters for databases is often a non-trivial task that if not managed properly can cause performance or stability issues.

Despite being cloud-based, multiple databases also present operational challenges. One such challenge is related to managing a large number of cloud services in your organization. With each application being served by many databases, your operations teams have to deal with more different pieces of software. Cloud services are easier to maintain than on-premises services, but the operational maintenance remains variable and non-zero. For example, some databases do seamless upgrades while others require intervention. Some databases need specific configuration for your context, while others can run perfectly on defaults.

Latency and scaling

In addition to the overall complexity impact of having multiple databases per application, latency is also impacted. In addition, it can be hard to get a handle on overall latency in a complex system. If a single operation of an application has to touch multiple databases, then the cumulative latency of all the databases and the application will affect the application's overall performance. Whereas a single database platform can perform many operations in a single request, in the cloud, each operation could be on a separate database that adds its own round-trip latency to the application and back. Indeed, synchronous operations may require many round-trip network hops (compounding latency) while others may be able to be parallelized, so all network hops occur at the same time (in this case, overall latency would be the highest latency of the group).

Scaling is another consideration. Each database supporting the application may scale differently, yet the application will perform only as well as its weakest link. Even with optimized infrastructure for each database, the operational strategy is complex, as performance hits plateaus on one database or another as you scale.

"Each database requires developers to learn not only the conceptual basis of the database and model, but also the accompanying libraries."

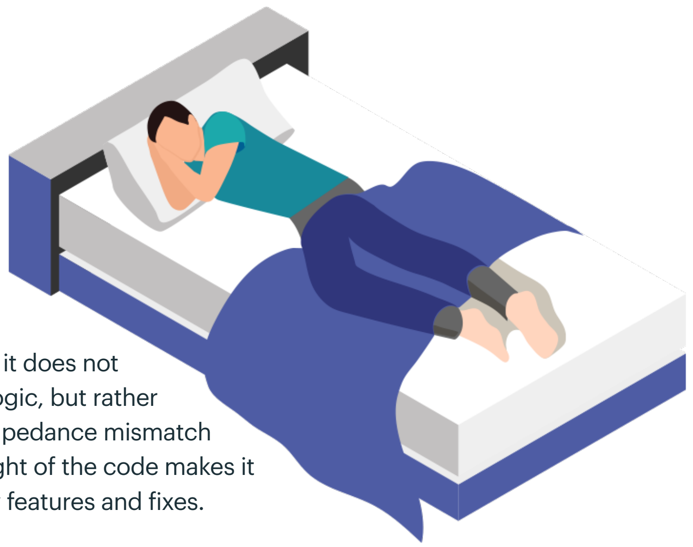
Beware the short blanket

Think about trying to make a bed with a blanket that is not long enough to cover the mattress. Tucking the blanket into the foot of the bed leaves the head exposed. Lining it up at the head leaves the foot exposed. Laying the blanket in the middle exposes both the head and the foot. All the options are inadequate.

Similarly, the old-line database providers give unified organizational deployment and scaling, but their ill-fitting models must be remedied by additional developmental complexity (see *lower left*). Cloud vendors provide an array of purpose-built database types, which unfortunately breeds operational complexity in having to connect, scale, and run multiple databases per application (see *lower right*).

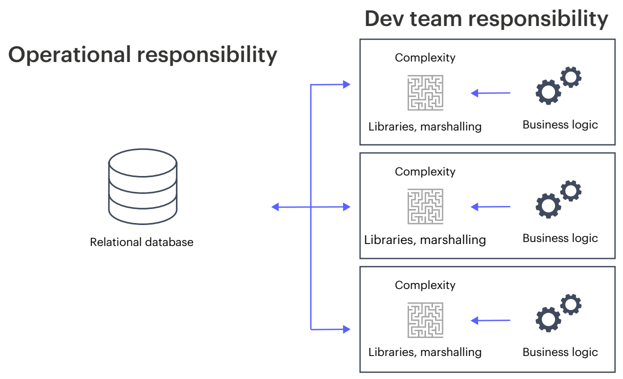
Ultimately, both approaches are inadequate. Moving the complexity to development or operational teams does not reduce the overall complexity. The massive engineering effort required to move an application from a single data model in an old-line database to many purpose-built databases in the cloud could actually be a waste of time if it doesn't actually reduce complexity but merely reassigns the complexity to a different team.

Even worse, migrating an application to the cloud can usher in an endless cycle of refactoring. For example, consider an application that is running well and stable on a variety of data models, but is shoehorned in a relational database. While the application may run well, its underlying code is lengthy and hard to maintain—much of it does not address business logic, but rather object-relational impedance mismatch problems. The weight of the code makes it difficult to add new features and fixes.

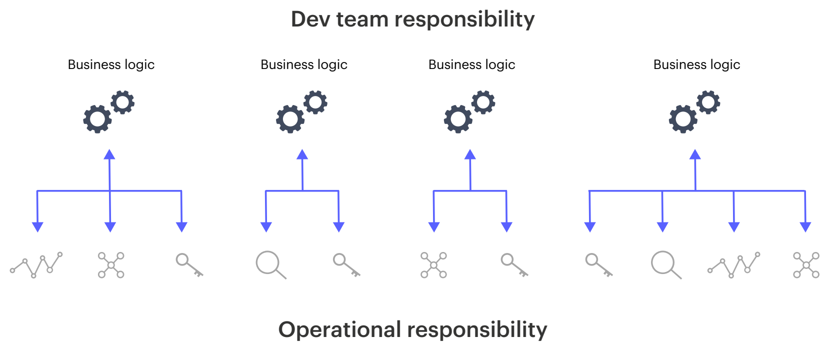


To solve this problem, the development team sees, say, a graph database, a time-series database, and a search database that would eliminate the need for large parts of the existing codebase. Switching to these purpose-built databases seems reasonable. Re-engineering occurs and the next release of the application is written to talk to these three new databases.

The development team quickly iterates on new features... but the production environment is now slow and unstable. The operations team sees issues with uptime and struggles to deal with troubleshooting multiple new databases. It becomes difficult to diagnose the exact problem as the cloud-provided solutions are black boxes that supply limited visibility into the problematic databases. The operations team starts to advocate moving to another database strategy (most likely back to a relational database) that is more stable and performant. The cycle of pain and expense begins again as you move the blanket around on a bed it simply doesn't fit.



The complexity stays on the side of the development team as they have to map relational data to reflect the true model of the application (business logic).



The complexity of the application is operational in connecting to and maintaining multiple, purpose-built databases as advocated by cloud vendors.

"The strategic option is to find a path that incorporates the best of both worlds. A path that allows multiple, native data models to co-exist yet scale predictably."

Strategic flexibility with Redis Enterprise

So far, this document has addressed only two options, inadequate as they might be. Both the RDBMS-for-everything and the multiple-database-models-in-the-cloud approaches work only on a tactical level, solving one problem in a regimented manner, shifting difficulties around the organization, and ultimately setting up the organization for cyclical, expensive re-engineering projects.

A more strategic option is to find a path that incorporates the best of both worlds. A path that allows multiple, native data models to co-exist yet scale predictably. A path that lets developers use one library and connect to one endpoint, and have access to multiple models. This strategic, flexible path exists—and it leads to Redis Enterprise.

Redis Enterprise is a database platform built around a core of Redis. Redis, without any extensions, provides a key-value data access model with the addition of data structures like Hash maps, Lists, Sorted Sets, streams, and more. Redis can be extended with modules that introduce entirely new functionality to the database. These modules implement native data models such as graph, full-text search, document storage, and time series, as well as additional functionality around artificial intelligence and event-driven scripting.

The advantages of the Redis Enterprise approach boils down to three key factors:

1. One operational interface, with multiple data models
2. A simple, unified interface
3. Cloud-born and bred

One operational interface, with multiple data models

Redis Enterprise provides the best of both worlds—it offers a single operational interface where many data models can coexist on a single data plane. Redis Enterprise was developed out of the need for enterprise-scale users to operationalize their Redis instances. The Redis ecosystem evolved a robust, pluggable module system that hooks deeply into the heart of Redis, allowing Redis to move beyond the built-in key/value model and host other data models. This multi-model approach lets it inherit the operational characteristics of Redis while adding powerful new capabilities. As you scale, you need only add more infrastructure to Redis Enterprise.

Enabling a module in Redis Enterprise is as simple as clicking a button in a UI, but it drastically changes how data is stored and queried. Redis has produced a number of modules that enable a number of different data models, including:

Search and Query
Full-text search and secondary indexing

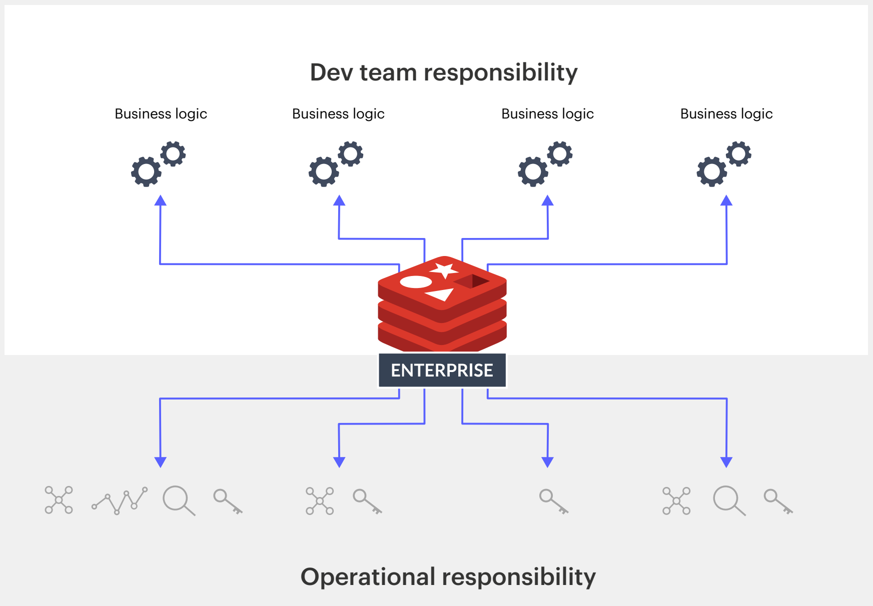
JSON
JSON document store

Time Series
Record data samples in high volumes and aggregate and downsample on the fly

Modules are optional, you can turn on only the ones you need for a specific project. Redis Enterprise is multi-tenant, so you can create databases for individual needs with specific modules or you can create databases with many modules working in concert, allowing you to zoom in on the specific capabilities you need for your application.

These modules, alongside Redis' built-in key-value data structure, enable a wide range of uses. It's important to understand that these modules are not capabilities built on top of the key-value model, but rather models that are accessing the internals of Redis to store data in memory and persist it to disk. Many of these models can be enabled on the same database, so it is possible, for example, to use full-text search over graph nodes or store JSON

"Redis Enterprise provides the best of both worlds—it offers a single operational interface where many data models can coexist on a single data plane."



Redis Enterprise enables an elegant separation of development and operational concerns without compromising on flexibility of the data model.

“ Redis Enterprise is cloud agnostic. Redis is not a cloud vendor, so you can choose the cloud and deployment options—in the cloud or on-premises—that work best for your organization and applications. ”

documents alongside time-series data. Because the different data models exist on the same data plane, even transactions are possible between models.

A simplified, unified interface

Because one Redis database can serve multiple models from the same data plane, it becomes a single point of connection. All of the connection properties are reduced into one and there are no performance penalties or extra latency for using multiple, native data models. Additionally, developers need to understand only one programmatic interface for all the different data models—it all boils down into the Redis protocol. In effect, you gain the benefits of using separate databases in one operational package.

Cloud-born and bred

Redis has always been a solution for the cloud. Redis Enterprise extends this with an overall package that is aware of the cloud environment with features like zone and rack awareness, monitoring, and bifurcated data and administrative planes.

Since Redis uses system memory for primary data storage, it creates an operational dynamic well suited to running on any cloud or multiple clouds. Redis Enterprise also supports Redis on Flash, which enables SSD-based storage on cloud instances to act as an extension to DRAM, enabling high performance of very large datasets on the same infrastructure at an affordable cost. As an in-memory database, Redis is ready for the next generations of data storage as the industry evolves from spinning disks and block-based Flash storage to byte-addressable durable storage.

Significantly, Redis Enterprise is cloud agnostic. Redis is not a cloud vendor, so you can choose the cloud that works best for your organization and applications. This mitigates cloud lock-in and enables hybrid cloud and multi-cloud deployments. Additionally, Redis provides a variety of deployment methods: choices range from a fully managed cloud environment to licensed on-premises deployments.

Redis Enterprise: a strategic solution

Solving a single problem is often relatively uncomplicated. Making choices that solve multiple problems is typically more challenging. Add in the need to solve multiple current problems while also addressing multiple problems for the future, and things can get extremely complex. This is the heart of strategic problem solving. Database choices are sometimes made lightly, other times they're predetermined before a project even commences. Either way, that is not a strategic choice.

Redis Enterprise provides a new path that can strategically solve both development and operational problems for not just today, but also for the foreseeable future. It soothes the pain of data models in development as well as performance and stability in operations at the same time, helping ensure a decision that will stand the test of time.

Freedom to bring your own data model

Applications built with Redis Enterprise work better because they store data the way your application uses it. Redis Enterprise supports a variety of built-in, rich data models that can be used to match the task at hand. As complete implementations, Redis Enterprise doesn't try to shove data into inappropriate data metaphors, which helps you reduce application-level code and complexity. Reducing your application-level code and complexity results in more-productive developers focused on your business problems, not trying to cram data into an inappropriate data model.

Built-in operational flexibility

Since Redis Enterprise enables many data models to coexist within the same cluster or even the same data space, it eliminates connections between different, independent databases.

"Redis Enterprise provides a new path that can strategically solve both development and operational problems for not just today, but also for the foreseeable future."

“Database TCO should take into account not just the price per time unit, but also the organizational impact. Your database should be flexible enough to natively support more than a single data model, optimizing and future-proofing the data layer.”

Consolidating multiple models under one connection reduces the brittleness and instability brought on by multiple connections. In Redis Enterprise, you can even run transactions between data models, something just not possible with a collection of independent databases.

No cloud lock-in

Because it doesn’t come from a cloud vendor, **Redis Enterprise isn’t locked to a specific cloud—you can take your data to an entirely new cloud should the need arise, or even deploy in a hybrid or multi-cloud configuration.** You have full control over the infrastructure running Redis Enterprise, you can easily add more infrastructure to support a cluster. Inside the cluster, you can create isolated or combined databases that have purpose-built data models.

Infrastructure flexibility

The total cost of ownership of a database should take into account not just the price per time unit, but also the organizational impact. The solution you pick should be flexible enough to natively support more than a single data model, optimizing and future-proofing the data layer against unanticipated shifts and changes. **Redis Enterprise simplifies data-layer challenges by giving developers and architects freedom and flexibility to build applications that deliver performance without sacrificing operational efficiency.** This balance advances the interests of both those writing software and those charged with running it, minimizing undue burden on any part of the organization.

Both the old-line database vendors and the cloud providers possess soapboxes from which they loudly proclaim that they have the true pathway and that the other is completely wrong. Both sides have reasonable critiques of data-modeling weaknesses and operational complexity against one another. Thankfully, the choice is not binary: you have options outside this false dichotomy. You can learn more about the advantages of Redis Enterprise at redis.com/modules.

redis

About Redis

Modern businesses depend on the power of real-time data. With Redis, organizations deliver instant experiences in a highly reliable and scalable manner.

Redis is the world's most popular in-memory database, and commercial provider of Redis Enterprise, which delivers superior performance, matchless reliability, and unparalleled flexibility for personalization, machine learning, IoT, search, e-commerce, social, and metering solutions worldwide.

Redis, consistently ranked as a leader in top analyst reports on NoSQL, in-memory databases, operational databases, and database-as-a-service (DBaaS), is trusted by more than 7,400 enterprise customers, including five Fortune 10 companies, three of the four credit card issuers, three of the top five communication companies, three of the top five healthcare companies, six of the top eight technology companies, and four of the top seven retailers.

Redis Enterprise, available as a service in public and private clouds, as downloadable software, in containers, and for hybrid cloud/on-premises deployments, powers popular Redis use cases such as high-speed transactions, job and queue management, user session stores, real time data ingest, notifications, content caching, and time-series data.

Follow us:

redis.com