Couchbase

GUIDE

Database Advice Guide

Developer's Guidebook

Contents

INTRODUCTION

This paper will cover:

- Areas to consider when choosing a database platform
- Setting up and configuring your database
- Ease of use
- Production performance, high availability, and scalability
- Security and data protection

The database is one of the most important parts of a software system, whether for an application or a data warehouse project. As such, this document offers a virtual checklist that you can use when evaluating a database.

Don't underestimate the work that you need to spend in planning your data needs and scope. Picking a database is a long-term commitment. The database is the foundation of your application and provides secure, reliable storage and access to all of your information. Without trustworthy data, you don't have much of an application. This paper will focus on operational databases, those technologies designed as the data storage and data access support for application or microservices development. These databases have different attributes than pure analytical data warehouses, data marts, and data lakes, such as Snowflake or Databricks, which are used to aggregate data from a wider range of sources, requiring some extraction, transformation, and load (ETL) processes to bring data into the database. Additionally, the queries are often much more analysis-based than in an operational database.

CHOOSING A DATABASE PLATFORM

RELATIONAL DATABASES OR RELATIONAL DATABASE MANAGEMENT SYSTEMS (RDBMS) REPRESENT AROUND 80% OF THE OPERATIONAL DATABASE MARKET.

There are hundreds of databases and data-related platforms on the market. There are many ways to narrow down the scope and the following sections examine some of the most immediate choices.

The first question you will likely need to answer is, "Do I want to use a relational database?" And if so, what are the features I think I need in a relational database versus what I can have with a NoSQL database? Relational databases or relational database management systems (RDBMS) represent around 80% of the operational database market.

RELATIONAL DATABASES

A RDBMS is structured on the relational model of data that organizes information into tables of rows and columns, that are related to each other, and usually have a unique key for each row. Typically, different entity types (e.g., product or region) that are described in a database have their own table with the rows representing instances of that type of entity and the columns representing values attributed to that instance. Each row in a table has its own specific key and rows can be linked to rows in other tables by storing the unique key of the target row ("foreign key"). The linking of tables together allows for data to be set up in a way where data is not duplicated in the database, making storage very efficient. In order to get the best of a relational database, its schema (table and relational layout) is planned well in advance and tends to be difficult to change without significant changes to the application. This design rose to popularity from the '70s through the '90s when storage was very costly.

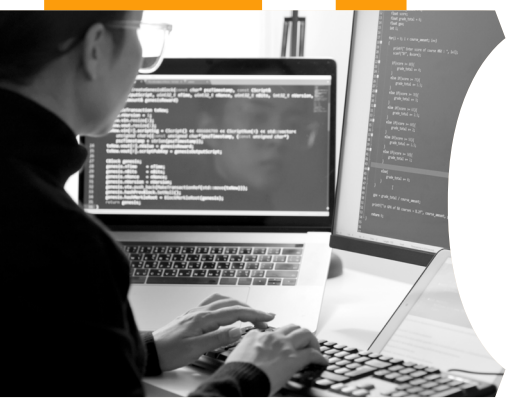
Nearly all RDBMSs use SQL (Structured Query Language) as the language for querying and updating the database. The SQL language had two big advantages over older APIs. First, it gave rise to the idea of accessing many records with one single command, and second, it eliminated the need to specify how to reach a record. It is essentially a declarative language, but with procedural elements. With the combination of organized table structure design and an easy-to-use query language, relational databases became popular due to their simplicity, robustness, transactional performance, and compatibility in managing data with other systems. For example, a fairly simple SQL SELECT statement could have many potential query execution paths. The RDBMS determines the best "execution plan" using features such as a cost-based optimizer to choose the correct indexes and paths.

The challenge for the relational database comes typically in two main areas: flexibility and scalability. As previously mentioned, the schema design of the database is often developed in the early stages of application development, with table design and key relationships intended to stay fairly static. Unfortunately, as the needs of the application change or as the desire to add new features evolves (often due to business changes) there is a need to redesign the schema. This often requires analysis to see the knock-on effects of the change and it may involve a database administrator (DBA) and other parts of an organization. The other key challenge is scalability. Relational databases scale up well on a single machine but work less effectively when scaling out across multiple servers that can distribute the load. During attempts to scale to hundreds or thousands of servers, the complexities can become overwhelming. The characteristics that make relational databases so appealing are the very same that also drastically reduce their viability as platforms for large distributed systems.

Examples of relational databases include:

- Amazon Aurora
- IBM Db2
- Microsoft SQL Server
- MySQL
- Oracle
- Postgres

Relational Database Pros	Relational Database Cons
• Strong support for data integrity and transactions • Highly functional/popular query language including "joins" between tables • Powerful indexing capabilities, query planning, and cost-based optimization	• Rigid structure slows ongoing application evolution – less capable with semi-structured data • Scalability challenges and high costs • Overly efficient schema design can result in many tables and joins, which can impact read and write speeds



NOSQL DATABASES

Non-relational databases, often referred to as NoSQL databases, are designed to store and retrieve data that are not stored in relational table format. The most common types of NoSQL databases are key-value store, wide columnar, document, graph, and time-series.

- **Key-value store:** The simplest form of a NoSQL database, data elements are stored in key-value pairs that can be retrieved by using a unique key for each element. Values can be simple data types like strings/numbers or complex objects. Great speed can be achieved via key-value access.
- **Document databases:** Store data in JSON, BSON, or XML documents, in a form that is much closer to the data objects used in applications. This means less translation is required to use the data in the applications (compared to relational databases). Collections are a grouping of documents used to help organize information. Documents provide great flexibility to change the database as the application evolves, since there is no schema enforced by the database.
- **Graph databases:** Focusing on the relationship between the elements, data is stored in the form of nodes. Connections between nodes are called edges. The goal is to be able to easily identify relationships between data by traversing the links.
- **Time-series databases:** Operate on data that is evaluated at regular intervals such as stock feeds or operating system activity logs. Here, the ability to zoom in and out at different granularity levels (from minutes to days, for example) may reveal trends from the data.
- **Wide-column databases:** Organize data into column families rather than traditional rows, allowing efficient access to sparse or large datasets. They're optimized for high write throughput and analytical queries across large volumes of data, making them well-suited for time-series, IoT, and log data.

OVERALL, NOSQL DATABASES ARE DESIGNED TO GIVE DEVELOPERS MORE FLEXIBILITY IN HOW THEY STORE AND RETRIEVE DATA.

Overall, NoSQL databases are designed to give developers more flexibility in how they store and retrieve data. These databases are sometimes architected to improve horizontal scaling with distributed architectures. Nodes can be added and changed without requiring changes to the application, with data automatically replicated to new nodes. This also often results in better uptime with less work required. These systems were designed for big data and modern development practices like agile development, CI/CD, and serverless. Some NoSQL databases have proprietary query languages, while others have adopted SQL. Advanced databases offer sophisticated indexing technologies, ACID transaction support, and cost-based optimizers. It's difficult to speak on the pros/cons of NoSQL as a whole, because there is so much variety under that umbrella.

Examples of NoSQL databases include:

- Cassandra
- Couchbase
- Cosmos DB
- DynamoDB
- MongoDB™
- Redis
- Neo4j

NoSQL Database Pros	NoSQL Database Cons
• Flexible schemas/schemaless • High availability with less maintenance • Automated scaling • Versatile/specialized data modeling	• Potentially limited indexing (depends on type) • Less data integrity and consistency is possible • Less familiar to the industry • Data duplication potential

PERFORMANCE

Performance is another important area to consider when choosing your database. Almost all databases will work quickly at a small scale with limited concurrency, numbers of nodes, data, and users. While you may not have full clarity on the expected growth of your application and database in the future, the more you know the better you can select the right database. Two typical measures of performance are throughput (operations per second) and latency, which measure the round-trip time between a client and the database. For NoSQL databases, the industry standard for measuring performance is the Yahoo! Cloud Serving Benchmark (YCSB). A neutral third party, benchANT, publishes performance results from several products and vendors.

DATA ACCESS

Some databases are very limited in how one can access data, while others provide a variety of ways to get at the data. Key-value APIs use gets and puts for simpler and often faster data retrieval. Textsearch queries (using facets, fuzzy, language awareness, scoring, etc.), also known as full-text search, can be a very useful database feature, providing search functionality for users within applications. Another option rising in popularity is vector search, enabling search on semantic meanings rather than exactly values. SQL is an industry standard for relational databases, but also has support in many NoSQL databases. Depending on the combination of your present and future needs, ensure your database has your querying needs covered. Having many query options makes a database multi-model and lets developers do more in one database, which can reduce vendor and data sprawl. Read "A Story of How Multi-Model Databases Can Reduce Data Sprawl" for more detail.

CLOUD STRATEGY

Moving your database to the cloud can offer flexibility, affordability, and scalability, but there are many ways to run in the cloud. Some organizations may prefer to self-manage in the cloud, giving them full control over their configuration and data, yet still getting the benefits of scalable infrastructure. Additionally, many look to automate self-management in some way using containers and Kubernetes. This makes it easier to administer a database once it is up and running and over time, and provides a standard API that can be used on a variety of cloud providers. For those looking for even less management burden, many database vendors now offer Database-as-a-Service (DBaaS) solutions. These let users provision clusters in a cloud provider of their choice and leave the majority of administration up to the database

vendor. For all of these choices, a critical decision is the balance of control versus convenience. Often the more convenient the system, the less control a user has over what happens in the database. Most of the large cloud infrastructure providers also offer a host of database services. While many start with those databases due to ease of access, you need to consider the issue of vendor lock-in and the benefits of a multicloud strategy, like pricing flexibility and system resiliency.

DATABASE-AS-A-SERVICE

A Database-as-a-Service typically runs on a cloud computing platform and is fully managed as-a-service by a database provider. Database services take care of replication, scalability, high availability, upgrades, and patching, making the underlying software more transparent to the user. Some databases provide very specific capabilities, while others are more broad and flexible. Cloud providers like AWS, Google Cloud, and Microsoft Azure offer a variety of DBaaS offerings, each focused on narrow capabilities. Sometimes that selection can add complication and sprawl.

MOBILE USAGE

Will your DBaaS need to sync data to a mobile or edge application? If so, you should consider the options for an embeddable database that can be synced with a central clustered database. This brings the advantages of always-on apps regardless of web connectivity and speed. With a smart syncing and storage strategy, users will be able to quickly and easily search, update, and analyze data at the edge. Ensure that your database has a mobile strategy to enable sync, offline-first, and/or resource-constrained environments.

ANALYTICS

As stated at the beginning, the focus of this paper is operational databases and not pure analytical databases, which doesn't mean that no analysis happens in an operational environment. Analysis is of course a very broad term. Analysis performed on the data tied to an operational dataset is often only used for one application. And by not having to move the data to a data warehouse, users can examine data in near-real time. The industry refers to this as hybrid transaction/analytical processing (HTAP) or hybrid operational and analytical processing (HOAP). It "breaks the wall" between transaction processing and analytics, and enables more informed and faster business decision-making. To help ensure that queries run quickly at scale, some databases accelerate queries by utilizing a massively parallel processing (MPP) engine and/or columnar storage.

COST

While most databases offer some sort of free tier or community edition, it is often not enough to support a production application. In planning for application deployment, you need to forecast your data volume needs, identify the services needed to meet your functional requirements, and understand the performance capabilities to meet your goals. Most vendors provide detailed pricing information on their websites. Conducting a proof of concept (PoC) is a good starting point to evaluate pricing. Understanding scalability also helps you identify the sizing needs for scale to determine the overall cost in production.

SETTING UP AND CONFIGURING YOUR DATABASE

Learning your database platform

EASY, QUICK SETUP

One of the keys to utilizing database management systems is a fast, simple setup. The traditional steps of procuring hardware, installing software, testing that the setup was performed suitably, etc., are not feasible in the modern agile development life cycle. For many organizations the solution is to move to a DBaaS model.

Database-as-a-Service is a managed service that lets users access database services without having to be concerned about managing infrastructure or software updates. It is the easiest and fastest way to deploy a modern database. Basically, a fully managed cloud service heavily reduces your database management burden. A database cluster can be deployed in just a few clicks. All that is required is to create an account to quickly generate clusters and databases, import data, and utilize the available database services of a fully managed DBaaS in a cloud environment.

INTUITIVE USER INTERFACE

An intuitive interface works the way the user expects; allowing users to navigate the UI instinctively while focusing on the tasks rather than struggling to navigate a complex interface. While you may want to script common tasks and/or use an API to manage your database, using a UI to explore and visualize data can be very helpful.

THE MOST POPULAR LANGUAGE FOR INTERACTING WITH DATA IS SQL.

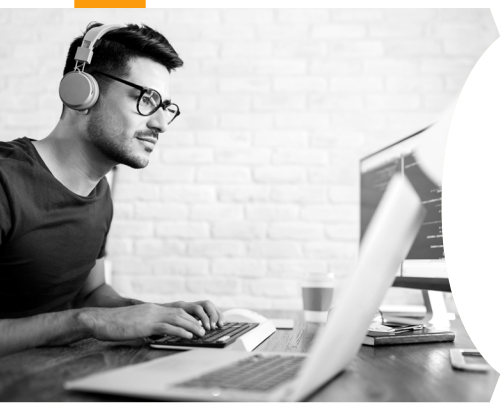
FAMILIAR SQL SYNTAX SUPPORT

The most popular language for interacting with data is SQL. SQL is a language that has been around in relational databases since the 1970s, and has expanded to SQL++ for use with non-relational JSON document databases. Whether you're looking at relational or non-relational databases, robust SQL support means that many database and query skills will translate from database to database, and will help your team to ramp up on a new database faster without needing to hire experts in a proprietary query language.

Data modeling

FLEXIBILITY

Data flexibility is an important capability that allows teams to more quickly and efficiently respond to changing requirements. Document databases offer flexible JSON models with the schema enforced by the application instead of the database. Users can model data in a way that fits their application objects, nest documents, or even emulate RDBMS models. JSON provides a simple, lightweight, human-readable notation. It supports basic data types, such as numbers and strings; and complex types, such as embedded documents and arrays. JSON provides rapid serialization and deserialization and is a very popular REST API return data type.



DENORMALIZING RDBMS DATA

One of the many advantages of using a document-based database is the ability to use a flexible data model to store data without the constraints of a rigid, predetermined schema. When moving from an RDBMS data model to a NoSQL document model a common practice is to denormalize the data. Main RDBMS tables and auxiliary tables are often combined as nested parent and child JSON documents. However, it is also beneficial to consider how the data will be accessed since this greatly influences the data modeling strategy. For instance, smaller documents can be read/written faster. Please note that denormalizing data may increase the duplication of data and increase storage size. Storage costs have continued to decrease every year, but still should be a consideration.

NESTED JSON DOCUMENTS

Nested JSON documents minimize the need for JOINs which makes data access faster and more efficient. It can also simplify the data model. A typical RDBMS database architecture diagram has many more table objects than a NoSQL architecture diagram has corresponding collections. But it is important to note that with consolidation, documents may be larger. If your model gravitates toward large documents, it's important to make use of sub-document APIs when possible (APIs that allow partial reading/writing of documents).

JOIN SUPPORT

In the NoSQL document database world, SQL is uncommon, and JOINs are more so (but not completely absent). Just as in a relational model, a JOIN clause can be used to link two or more source objects. Look for the following types of joins, and if they are supported:

- ANSI JOIN (joining between multiple documents on an arbitrary field)
- Lookup JOIN (joining between multiple documents based on their ID or document key joined with an arbitrary field)
- Index JOIN (the reverse of a lookup JOIN, requiring an index)

When exploring a *distributed* document database, JOIN support across shards/partitions also needs to be considered, as they aren't always supported.

RDBMS CONCEPTS

Key concepts such as tables are widely understood. Many of these concepts can be mapped to a (roughly) equivalent feature in document databases.

RDBMS	Document DBMS
Schema	Scope
Table	Collection
Row	Document
Primary key	Document key
Index	Index

ACID TRANSACTIONS

Relational databases must support ACID transactions, due to the nature of their data modeling requirements. ACID transactions are required when updating multiple data in multiple tables: the operations must all succeed or all fail. JSON document databases initially skipped ACID transactions in favor of performance, but many of these NoSQL databases now support ACID transactions. With the SQL++ query language, multi-document ACID transactions can be achieved and use the common syntax of BEGIN, COMMIT, and ROLLBACK. Again, there may be limitations based on sharding/partitioning, so if your application needs to perform transactions between shards, check to make sure what support is available (if any).

Ease of development

Familiarity, industry standards, and good documentation all help in ramping up development efforts to accelerate developer productivity.

CRUD

A common way to get familiar with a database and its SDK is by writing a CRUD application. CRUD stands for:

- Create
- Read
- Update
- Delete

These are key developer actions that also serve as very useful coding examples. The time it takes to get a CRUD example up and running can help to familiarize you with the basics, and give you a baseline to evaluate more complex operations.

DBAAS

A DBaaS solution means there is less maintenance required as compared to managing database software yourself. Users can sign up and start working without having to worry about software setup, installation, patches, or even system updates. This allows users to focus on utilizing the platform and not "yak shaving."

SUPPORTED ACCESS SERVICES

In a relational database, SQL is the only way to work with data. For NoSQL databases, there are many other services that could be available (including SQL++). Full-text search (browser-like), analytic queries, embedded mobile database syncing, database event triggers, key-value access, and in-memory cache performance enhancement are all examples of services that may be included in a database. You could take a "**polyglot persistence**" approach and use a different database for each service, but this multiplies the maintenance and integration work (more "yak shaving"). Even if you don't plan to use all these services from the beginning, choosing a database that has these features ready to go can save you time, and provide a form of "future-proofing." Not to mention this approach can minimize "database sprawl" which can turn into an architectural nightmare.

AI SERVICES

Integrated AI services allow developers to deploy and manage generative AI (GenAI) workflows alongside operational data. These services support tools, prompt orchestration, and model invocation, providing a foundation for retrieval-augmented generation (RAG) and agent-based apps. The ability to host models close to the data improves performance, costs, and privacy.

Data access

MULTI-MODEL DATA ACCESS

A "multi-model" database is one that can support multiple access methods upon the same pool of data. The system provides unified data management, access, and governance, among other key features. For example, Couchbase provides the following ways of interacting with data:

- **Key-value:** The ability to read/write data via a fast "key" lookup, given Couchbase's memory-first architecture, and is great for simple use cases and ultra-fast performance.
- **SQL++:** The world's most popular language for querying data; SQL syntax support is a database industry standard data access method.
- **Full-text search (FTS):** A text "search engine" for data that also supports geography-based searches, fuzzy search, faceting, etc.
- **Geospatial:** Search data based on geography and distance.
- **Time-series:** Store and query sequences of timestamped data.
- **Analytics:** Query data with complex, ad hoc SQL++ queries, in an isolated environment with multi-tenant support.
- **Eventing:** Developers can write JavaScript functions that respond to data change events.
- **Vector:** Semantic querying on vector embeddings, often used for AI use cases.

Access to more than one of these data access methods helps to minimize sprawl and includes the efficiency benefit of users not having to learn multiple data management systems and SDKs, patch and upgrade multiple systems, manage multiple licenses, etc. This in turn reduces costs, as well as application development time and time to production.

SDKS, BIG DATA CONNECTORS, AND RELATED TOOLS

Developers generally access data via specific language-based SDKs (or ODBC/JDBC connectors in some circumstances). While your team may be focused on one language, it's important to evaluate the available SDKs, should the need for other languages come up in the future. Here's a sample checklist of SDKs:

Server-Side SDKs	Mobile/Desktop SDKs
Java	Java (Android)
Scala	Java (Desktop)
.NET	MAUI (.NET)
C/C++	Swift
Node.js	Objective-C
PHP	C/C++ (embedded)
Python	
Ruby	
Go	
Kotlin	
Rust	
JDBC/ODBC	

Also check for available big data connectors and other tools for real-time analytics, streaming, data modeling, and search platforms. Some include:

- Kafka
- Spark
- Elasticsearch (ELK)
- CData
- Erwin
- Tableau
- Talend
- Power BI
- Apache Airflow
- Apache Camel

WHILE MANY OF THESE TASKS CAN BE ACCOMPLISHED WITH A UI, HAVING COMMAND LINE TOOLS AVAILABLE IS IMPORTANT AND CAN PROVIDE EFFICIENCY AND INTEGRATION WITH OTHER COMMAND LINE TOOLS THAT YOU ARE ALREADY USING.

Finally, there are many AI tools and connectors that are used to build AI, chat, RAG, agentic, etc., applications. Some include:

- Langflow
- LangChain/LangChain4j
- Haystack
- Gemini
- Vectorize
- Unstructured.io
- NVIDIA NIM
- LlamaIndex
- AWS Bedrock

Development tools

COMMAND LINE TOOLS

CLI tools can be helpful for developers, DBAs, and DevOps alike. Some tools to look for include:

- Exporting data
- Importing data
- Creating backups
- REPL for SQL or SQL++
- Data/index migration
- AI/LLM
- Management and administration
- Configuration and settings

While many of these tasks can be accomplished with a UI, having command line tools available is important and can provide efficiency and integration with other command line tools that you are already using.

DATABASE IDES/SQL EDITORS

There are numerous editors or database IDEs available. You may already have licenses for them, or be familiar with them through your organization. Some popular choices include:

- **JetBrains** – A family of IDEs designed for developers. It has an extensive ecosystem of plugins, including many database management and development plugins.
- **DbVisualizer** – A popular tool utilized by developers and DBAs across platforms.
- **SQL Server Management Studio (SSMS)** – A tool primarily used by SQL Server database developers but that can be connected to the Couchbase platform.
- **Visual Studio Code** – A popular cross-platform IDE with an extensive ecosystem of plugins, including database-related plugins.

These tools may provide native support and/or generic ODBC/JDBC support.

JSON FORMATTERS/VALIDATORS/PARSERS/HELPERS

JSON formatters and validators are helpful when working with JSON documents. Most of the IDEs listed above also feature JSON plugins.

There are also popular and easily accessible web-based JSON tools:

- JSON Parser
- JSON Lint
- JSON Editor Online

Production performance, high availability, and scalability

As critical projects move from development to production, the needs for stability and operational reliability become paramount. Developers needing these features of distributed systems look to NoSQL databases to deliver them. These enterprise production features provide the benefits of high availability, performance, and scalability.

HIGH AVAILABILITY

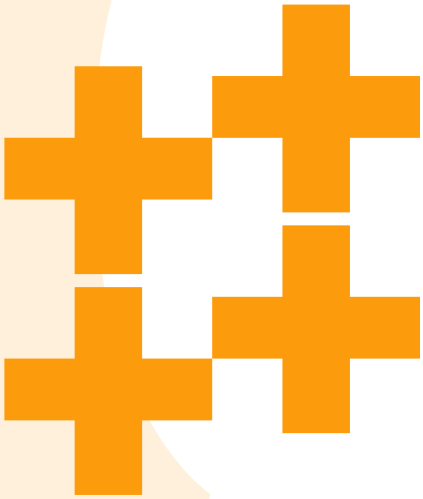
High availability (HA) and disaster recovery (DR) are two of the driving reasons to move from a traditional RDBMS to a NoSQL-based system. High availability, as a concept, allows operations to guarantee certain levels of availability even in the face of servers crashing.

NoSQL databases often support distributed database clustering, which helps deliver high availability. These databases also focus on the need for simple management and reducing interruptions to normal functioning. For example, operations can be done while the system remains online, without requiring modifications or interrupting running applications. All nodes do not need to be taken offline at the same time for routine maintenance such as software upgrades, index building, compaction, hardware refreshes, or other operations. Even provisioning new nodes or removing nodes can be done online without interruption to running applications, and without requiring developers to modify their applications.

Additionally, built-in fault tolerance mechanisms protect against downtime caused by arbitrary unplanned incidents, including server failures. Replication and failover are important mechanisms that increase system availability. For example, data can be replicated across multiple nodes to support failover scenarios. Ensuring that additional copies of the data are available is paramount to deal with the inevitable failures that large distributed systems are designed to recover from. This functionality is provided automatically without the need for manual intervention or downtime.

To deliver increased high availability, disaster recovery, and geographic load balancing, entire clusters can also be replicated to one or more alternate geographical locations. Often this creates additional challenges in linking clusters that span across a wide-area network (WAN) rather than simply extending local-cluster replication, but the benefits are often valuable to global applications.

For disaster recovery, customers can maintain a passive cluster (target) in addition to an active cluster (source) as part of their continuity plan. This is beneficial when the disasters occur at the data center or region level. Backup and recovery processes are essential components of disaster recovery and can also be distributed to reduce network overhead and related performance costs. The use of external blob/object storage services can help to store backup data close to the operational cluster for when it's needed.



SPEED AND PERFORMANCE

Anyone can have a superfast database if they spend millions on an extreme machine. But NoSQL users gain the benefits of scale-out – using many machines to share the load – using normal or moderate server offerings or cloud instances.

The power of a cluster of nodes allows operations to be distributed to handle workloads in an efficient manner. Additional nodes can be provisioned to help add specific capabilities to perform better (e.g., more data storage or increased memory allocations). This approach is called "multi-dimensional scaling."

As workloads increase, more nodes can be added without having to upgrade the existing nodes with more RAM, CPU, etc. (i.e., "vertical" scaling). Part of that distributed system design relies on a "shared nothing" architecture – each node manages its own resources including storage and processing. Additional nodes added for particular services, such as full-text search or big data analytics, can also use their own nodes without impacting performance of the others.

Replication and sharding are fundamental to automatically distributing data across nodes in a cluster. Thus, the database can grow horizontally to share load by adding more RAM, disk, and CPU capacity without increasing the burden on developers and administrators. This type of hardware efficiency is gained by employing asynchronous, non-blocking I/O for effective utilization of server resources. This helps increase both the storage I/O efficiency and the number of simultaneously connected clients per node.

Distributed database architectures help workloads be evenly distributed across cluster nodes, to reduce bottlenecks and allow users to take full advantage of the available hardware. The end result is higher performance, better utilization, and improved TCO (e.g., avoiding over-provisioning).

SCALABILITY

There are several ways to scale a database platform. Scaling is not only about increasing RAM or storage, but also about adjusting specific services that are available. If a cluster has, for example, a query service on one node but it is getting overutilized, then another similar node can be added or adjustment should be

made to increase RAM and CPU settings. A system should allow users to optimize services alignment to hardware over time, as rarely does the needs of an application grow linearly for all of the data services that it is using. During these optimization changes, the team should not have to worry about how to replicate data or queries to those services.

Various cluster configurations are used depending on the need for performance versus high availability, etc. For example, NoSQL clusters can implement peer-to-peer architectures, support active-active configurations, flexible topology including multi-master bidirectional ring topology, simplified administration, and filtered replication (where only certain data is replicated).

Decentralization of resources and parallelized streams (e.g., for replication) are essential to having a fully scalable platform. But scalable clusters are also reliant on having easy management when needing to expand the cluster. Keeping the cluster online reduces any downtime or service disruptions.

When adding more nodes to a cluster, the database needs to be able to duplicate data and services from one node before taking it online. NoSQL databases also use concepts such as rolling upgrades where individual nodes are updated without being taken offline. This can help when replacing a node, adding another node, scaling up a node with more resources, etc.

SECURITY AND DATA PROTECTION

IN CONSIDERING A DATABASE AND ITS IMPLEMENTATION, THE ONE BIG QUESTION IS OFTEN WHETHER THE DATABASE WILL BE SELF-MANAGED OR FULLY MANAGED BY THE DATABASE PROVIDER.

Security is an ongoing battle. In considering a database and its implementation, the one big question is often whether the database will be self-managed or fully managed by the database provider. Both will require many security considerations, but for Database-as-a-Service offerings, there are additional aspects that will need to be considered.

Core database security

ACCESS CONTROL

Access control is the foundation of database security, covering who can access the system (authentication) and what they're allowed to do (authorization). This applies to both users and applications seeking to access data. Two important security concepts are separation of duties and least privileged access.

- **Authentication**: Determines who is attempting to access the data. Users must be clearly and strongly authenticated. Database companies often use different models like:
  - **Password-based**: Built-in password authentication for both users and applications, where password strength policies are set based on complexity of the password, lifecycle, updating of the password, etc. Credentials transmission should be encrypted with transport-level security and/or hashed.

- **Certificate-based**: Certificates like X.509 provide an additional layer of security where the certificate authority validates identities and issues certificates.
- **Third-party/external authentication**: This allows you to plug into authentication platforms that organizations already have in place based on technologies like LDAP or Active Directory layer.

- **Authorization**: Once a user has been authenticated, authorization determines what a user or application is allowed to do with data. Sophisticated databases employ role-based access control (RBAC) where users are mapped to roles that determine the actions they are authorized to perform. Commonly, the roles between administrators/users and applications/data access are separated. Each user's roles can be as broad or as restrictive as required, from a full administrator having access to all administrative functions and data, to an analyst with read-only access to a limited dataset.

For on-premises, self-managed solutions, the entire security environment must be managed. There are many areas to consider, but a few key areas include:

- **Physical security**: Buildings, data centers, servers
- **Network security**: Firewalls, IP tables, WAN encryption
- **Operating system**: User management, security patches, and updates
- **Application**: Credentials
- **Key management**: Rotation, revocation, remediation

ENCRYPTION

Another critical aspect of privacy and data protection is encryption. The goal is to make sure that sensitive information is not made available in the case of unauthorized access. To do so, unencrypted data is encrypted by using an algorithm and an encryption key.

The following are important areas of data encryption:

- **Encryption-at-rest**: For data residing on physical media, encryption needs to protect against unauthorized access to the database files either from within the operating system or to the physical disks themselves. Often this is handled by the database working in conjunction with third-party software vendors which deny data access to anyone who does not possess an appropriate encryption key or is otherwise noncompliant with the configured security policy. Some examples of these vendors include Vormetric, Gemalto, Protegrity, and Amazon's encrypted EBS. DBaaS providers often include encryption-at-rest using the underlying cloud provider capabilities.
- **Encryption-in-transit**: Data often does not sit purely in a single database location. It is constantly on the move, being read, written, and replicated over networks. This movement requires an additional level of protection. Whether the data is moving between nodes in a single cluster within a data center, between data centers around the world, or out to mobile edge nodes, it is important to ensure that the database vendor provides robust encryption of data while in transit.

- **Field-level encryption**: This provides an extra layer of protection on specific values by encrypting user data before saving to the database itself. Not only is it encrypted over the network and on disk, but requires a separate key from the application to decrypt.
- **Encrypted backups**: Backups contain a large volume of data and providers should provide options for encrypting this aspect of your data management.

DATA RETENTION, SOVEREIGNTY, AND AUDITING

Protecting data is only part of the equation – many regulations also require organizations to show how that protection is being maintained. Policies vary by continent, country, and even state/region. The handling of government data brings in other rules and regulations.

- **Retention:** This allows administrators to set policies about how long data is kept, in what manner, and the expiration processes. Retention rules can be applied to entire databases or data subsets.
- **Sovereignty:** Needing to filter out data subsets for different regions is a very common requirement, but not necessarily an easy one when working with large datasets on a global scale. Sophisticated replication technologies are built into some database platforms that allow fine-grained controls and automated replication based on those controls.
- **Auditing and reporting:** In alignment with regulatory needs and business needs, organizations need to be able to audit their data and report out the status of data, both current and historical.

Hosted data security

If choosing to utilize a Database-as-a-Service, ensure that your DBaaS vendor provides sophisticated, multilayer security technologies and 24/7 monitoring. Systems should include things like private networking and encryption while data is both at rest and in flight. Often DBaaS vendors will provide detailed whitepapers on their Trust Center webpages.

Here are some areas to think about as related to security for data hosted with a database vendor:

- **Governance, risk, and compliance**: It is important to understand how a vendor's Infosec team maintains information security policies, risk management processes, and compliance with regulatory and industry standards relevant to information security. Often organizations will publish detailed policy descriptions and controls.
- **Corporate operational security**: DBaaS providers should have established operational requirements to support the achievement of service commitments to customers, relevant laws, and regulations. Some of the important operational areas include:
    - **Incident management**: Having policies and procedures in place in case incidents occur

    - **Vendor risk management**: To minimize risk in working with outside vendors
    - **Vulnerability management**: Policies for protecting infrastructure
    - **Endpoint security**: For example, antivirus detection for all devices
    - **Data classification and retention**: For example, mandating specific protections for different information types
- **Corporate network protection**: Providers must ensure networks and entry points are protected against unauthorized access and have mechanisms in place to prevent efforts like distributed denial of service attack (DDoS) protection. A DevOps team should be involved to proactive monitor systems, networks, hardware, and software within the environment.
- **Compliances/regulations**: Your hosted data should be protected in many layers. Vendors have a variety of ways to demonstrate their commitment to that protection. One of these is through meeting compliance standards. There are many standards, most of which are industry-specific. The most common of which is SOC 2 compliance.

As every organization and application has different security requirements, vendors may be able to meet the requirements even if particular compliance is not complete. It is best to engage with a vendor regarding security to ensure all your needs are met.

CONCLUSION

There are many factors like speed, flexibility, time to market, and costs to consider when choosing a database platform to ensure it aligns with your application requirements. Choosing the right database platform is not easy. Many enterprises turn to Couchbase to improve resiliency and performance, reduce data sprawl, and lower total cost of ownership. Today, 30% of the Fortune 100 manage critical data with Couchbase.

GETTING STARTED

Want to test-drive Couchbase? Sign up to try Capella, our Database-as-a-Service, for free. The service provides several sample datasets, data access tools, and tutorials to get you started. For those who are adventurous and want to jump right in, they can load their own data, connect an application, and start using features like SQL++ with various services such as Analytics, Full-Text Search, Vector Search, and Eventing. Mobile developers can explore Capella App Services, with an embeddable Couchbase Lite and sophisticated syncing technology.

ADDITIONAL INFORMATION

Here are several other ways to learn more about building with Couchbase:

- Tutorials and quickstarts
- Developer learning paths
- Couchbase integrations (frameworks, AI, connectors, tools, DevOps)

Couchbase

Modern customer experiences need a flexible database platform that can power applications spanning from cloud to edge and everything in between. Couchbase's mission is to simplify how developers and architects develop, deploy and run modern applications wherever they are. We have reimagined the database with our fast, flexible and affordable cloud database platform Capella, allowing organizations to quickly build applications that deliver premium experiences to their customers – all with best-in-class price performance. More than 30% of the Fortune 100 trust Couchbase to power their modern applications. For more information, visit www.couchbase.com and follow us on X (formerly Twitter) @couchbase.