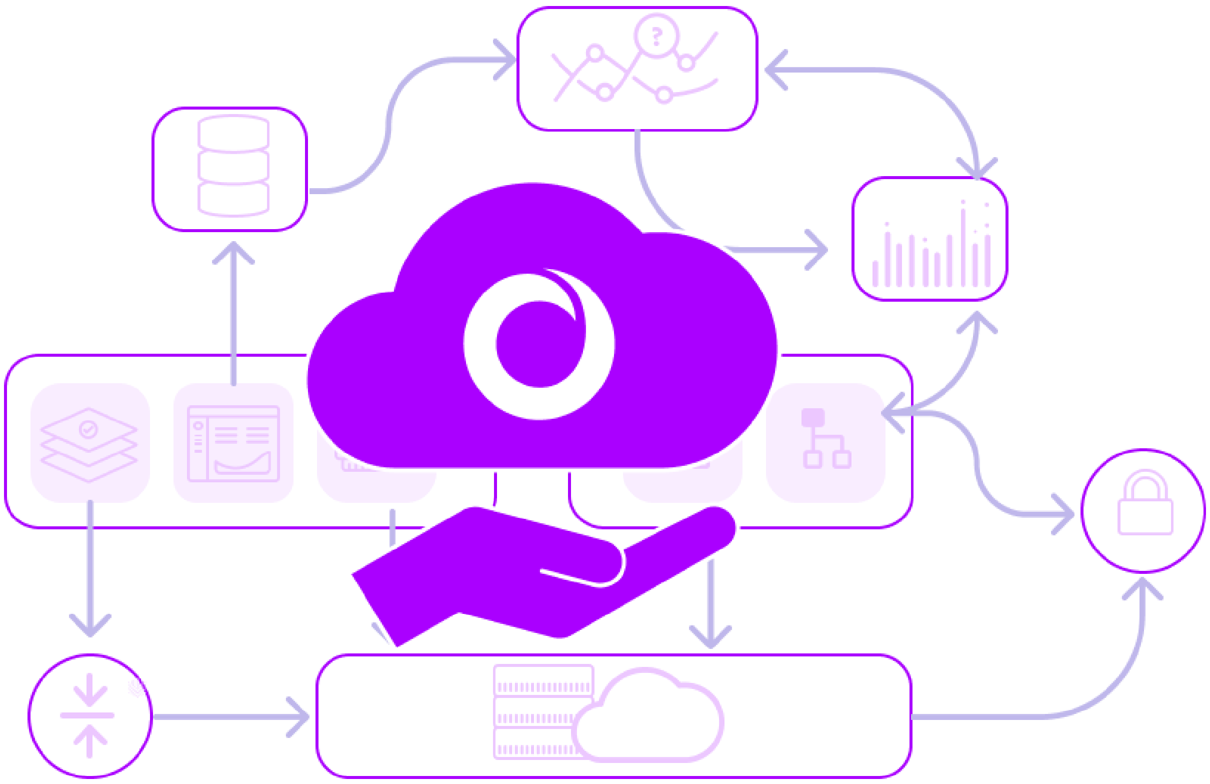
TECHNICAL WHITEPAPER

# Introduction to SingleStoreDB

Bobby Coates, Product Marketing Manager



2022

SingleStore

# Table of Contents

## About This White Paper

This white paper  is intended primarily for readers interested in data architecture, or looking for a better understanding of how SingleStoreDB works and is different from other databases. While SingleStoreDB can be deployed in the cloud, on-premises and a hybrid of both, this document focuses on the cloud deployed database-as-a-service offering.

## Current Challenges in the Database Market

The market is saturated with specialized database engines. As of January 2022, https://db-engines.com/en/ranking ranks over 350 different databases. There is value in specialty systems, but when applications are built as a complex web of different manually interconnected databases, a lot of that value is eroded. Developers are manually rebuilding the general-purpose databases via ETL and data flows between these specialized databases.

We believe two industry trends have driven this proliferation of new databases. The first trend is the shift to **cloud-native architectures** designed to take advantage of elastic cloud infrastructure. Cloud blob stores and block storage allow databases to tap into almost limitless, highly available and durable data storage. Elastic compute instances allow databases to bring more compute to bear at a moment's notice to deal with a complex query or a spike in throughput. The second trend is the demand from developers to **store more data and access it with lower latency and with higher throughput.** The performance and data capacity requirement is often combined with a desire for flexible data access. These access patterns are application-specific but can range from low-latency, high-throughput writes (including updates) for real-time data loading and deduplication, to efficient batch loading and complex analytical queries over the same data. Application developers have never demanded more from databases.

A common approach for tackling these requirements is to use a domain-specific database for different components of an application. In contrast, we believe it is possible to design a database that can take advantage of elastic cloud infrastructure while satisfying a breadth of requirements for transactional and analytical workloads. There are many benefits for users in having a single integrated, scalable database that handles several application types, including:

- Reduced training requirements for developers
- Reduced need to move and transform data
- Reduction in the number of copies of data that must be stored and resulting reduction in storage costs
- Reduced software license costs, and reduced hardware costs.

Furthermore, this enables modern workloads to provide interactive, real-time insights and decision making, enabling both high-throughput low-latency writes and complex analytical queries over constantly changing fresh data. Even more, it's done with end-to-end latency of seconds to sub-seconds from new data arriving to analytical results.

Adding incrementally more functionality to cover different use cases with a single distributed DBMS leverages existing fundamental qualities that any distributed data management system needs to provide. This yields more functionality per unit of engineering effort on the part of the vendor, contributing to lower net costs for the customer. For example, specialized scale-out systems for full-text search may need cluster management, transaction management, high availability and disaster recovery — just like a scale-out relational system requires. Some specialized systems may forgo some of these capabilities for expediency, compromising reliability.

At SingleStore we believe our design represents a good trade-off between efficiency and flexibility, and can help simplify application development by avoiding complex data pipelines. In this whitepaper we will discuss some of the key capabilities and uses for SingleStore.

## What is SingleStoreDB?

SingleStoreDB is the cloud-native database built with speed and scale to power data-intensive applications. SingleStore combines Hybrid Transactional/Analytical Processing (HTAP), ingest and query performance, scalability and resiliency through self-managed clusters while handling all installation-, operation- and management-related details. SingleStoreDB is designed and architected to tolerate and recover from failures. To maintain each cluster's high availability (HA), SingleStoreDB runs on a highly resilient cloud infrastructure with high availability built in.

SingleStoreDB high-level features include:

- Cloud-agnostic deployments on Amazon Web Services (AWS), Google Cloud Platform (GCP) and Microsoft Azure.

- Secure, with encryption of all data at rest, and TLS/SSL encryption for over-the-wire connections to the cluster
- Automated infrastructure provisioning, configuration, elastic scaling, backups, hardware maintenance and failure management, and upgrades
- Fully managed and resilient database-as-a-service with Universal Storage capabilities

### What is SingleStoreDB Not For?

SingleStoreDB excels at real-time and high throughput query use cases. It is a great database for data intensive applications, running both transactional and analytic workloads. However, there are use cases which SingleStoreDB is **not** designed to run. Some of these include:

- *Data Lake*. SingleStoreDB is not designed to be a data lake. It is designed for high-value data that is structured or semi-structured and ready to query. SingleStoreDB has open-source connectors for integrating with a variety of great object stores, including Amazon S3 and Hadoop File System (HDFS).
- *In-process database*. SingleStoreDB is not run as a library or in-process with an application. SingleStoreDB is a distributed database which runs in separate processes from the application, and applications connect to SingleStoreDB via a client driver.
- *Very small scale data volumes*.  Single host SQL databases can deliver excellent performance here because they don't need the infrastructure to scale.

## Core SingleStoreDB Technologies

### Architecture

SingleStoreDB is a horizontally partitioned, shared-nothing fully managed Database-as-a-Service which uses shared storage in the form of a blob store. A SingleStoreDB cluster is made up of Nodes, which hold partitions of data and are responsible for query processing. Each Node holds several partitions of data. Each partition is either a primary which can serve both reads and writes, or a replica which can only serve reads and is used for HA.

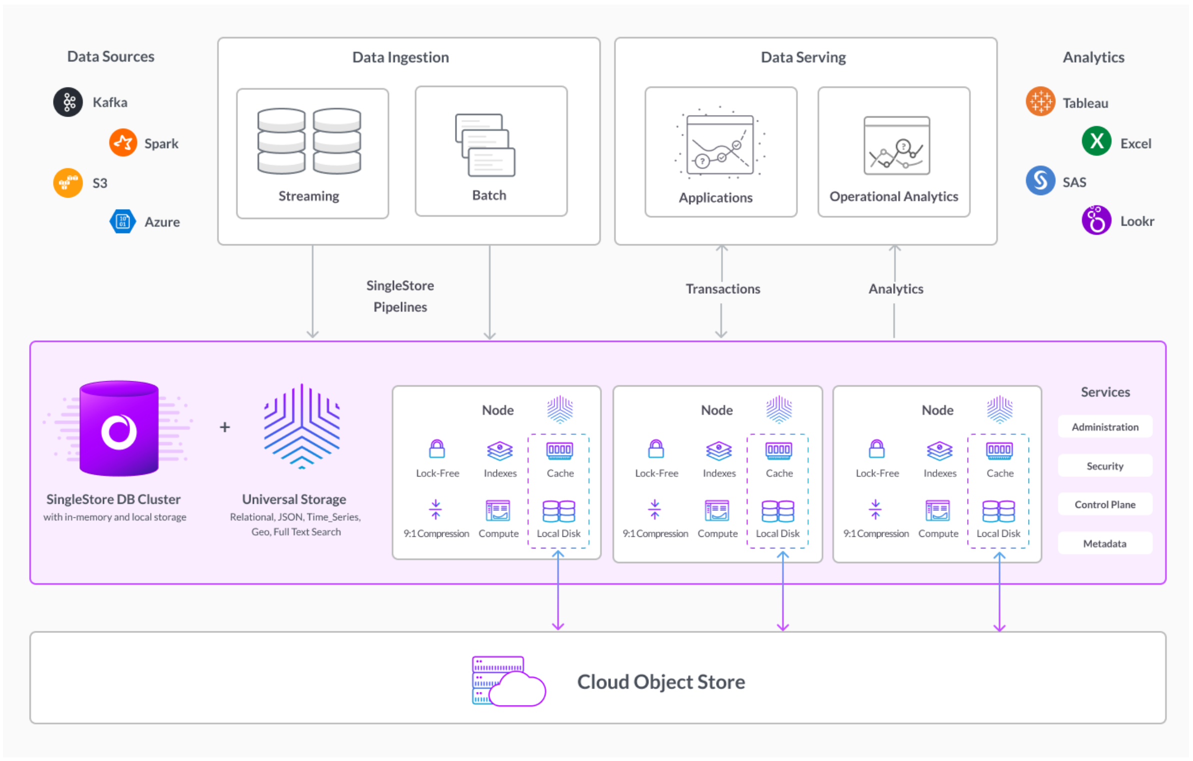


*Figure 1. Managed Services Architecture with SingleStore*

## Nodes

Nodes are responsible for storing data in partitions, moving data to and from the object store as needed, executing query requests against local partitions and returning collected data. The number of Nodes are configurable by architects and database administrators, since they are important to scale according to compute needs.

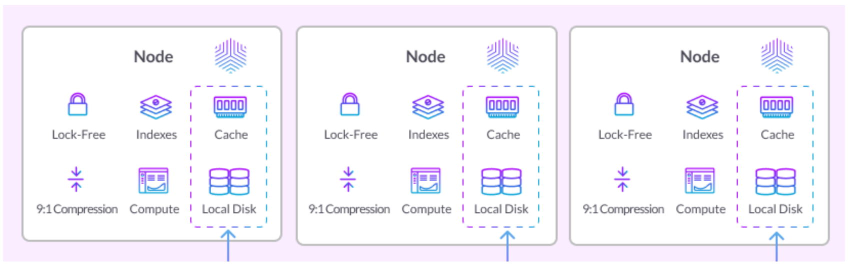


*Figure 2. Managed Services Architecture:Nodes with SingleStore*

## Universal Storage

As a storage engine built for HTAP, SingleStoreDB table storage needs to work well in a wide range of workloads. SingleStoreDB was originally designed with two separate table storage formats: an in-memory skiplist-based rowstore and a disk-based columnstore. SingleStoreDB table storage has evolved to a unified design, with the following goals:

### Efficient for both OLTP and OLAP access

Choosing between rowstore and columnstore table formats can be a hard decision for system architects. It requires the architect to know whether the access patterns of each table lean OLTP or OLAP when developing applications. This is especially true for workloads having both OLTP and OLAP aspects on the same tables, like real-time analytic use cases running analytics concurrently with high-concurrency point reads and writes. In the past, some customers created a rowstore table over the recent data to serve complex write workloads such as uniqueness enforcement, along with a columnstore table storing the older data for efficient analytics. This was hard to manage since it required the architect to implement processes to move rows as they age, and combine the results between tables during reads.

The primary goal of building the SingleStoreDB Universal Storage table is to provide a unified table type that works well both for OLTP and OLAP. This eliminates the burden on users to choose the data layout suitable for their particular workload. Furthermore, data layouts used to serve different access patterns should be integrated within one storage model, so the system doesn't pick up the disadvantages of different data layouts when trying to replicate data across them. Having one unified table type allows demanding HTAP workloads to work efficiently, without the complexity of managing data movements across tables serving different parts of the workload — or duplicating the entire dataset in multiple data layouts.

### Tiered storage

The storage model is optimized for tiered storage delivered by cloud providers. Writes to the blob storage (Cloud Object Store) are done as data files that are reasonably large in size (10s of MBs) to minimize the cost and amortize the latency of the requests. Data files are also immutable, allowing easy caching of the files on local storage which minimizes reads from the blob storage.

## Database Storage

SingleStoreDB stores data remotely in an object store (unlimited storage database) and locally in nodes (both on disk and in-memory). When you deploy a cluster, the databases will automatically use the remote object store without manual setup or configuration.

### Cloud Object Store

AWS, Azure and GCP clusters use cloud object store to store databases. The use of a cloud object store separates where data is stored (in an object store external to the SingleStoreDB compute) from where the data is processed (in a SingleStoreDB compute cluster). Because cloud object store databases are stored remotely, their size is not limited by the size of the persistent cluster cache, but rather only by available external object storage. On public cloud object stores, this is for all practical purposes unlimited.

In a cloud object store database:

- All columnstore data is stored externally in an object store. Additionally, recently accessed columnstore data objects are cached locally in the compute cluster's persistent cluster cache.
- All rowstore data is stored locally in compute cluster memory and externally in object storage.
- Data updates made on the cluster are flushed to the object store asynchronously. Typically, the object store will be no more than one minute behind the latest update on the cluster.

When you run the CREATE DATABASE command on AWS, Azure and GCP clusters, it automatically creates an unlimited storage database. BACKUP DATABASE and RESTORE DATABASE commands can be used for database backup and restore respectively. Data will be automatically restored into unlimited storage databases on AWS clusters.

# Query Code Generation

SingleStoreDB also supports full query code generation targeting LLVM through intermediate bytecode. LLVM compilation happens asynchronously while the query begins running via a bytecode interpreter. The compiled LLVM code is hotswapped in during query execution when compilation completes. Using native code generation to execute queries reduces the instructions needed to run a query compared to the more typical hand-built interpreters in other databases — it's worth the time to perfect it.

## Compiled Queries and Query Plan Caching

Steps for query execution, code compilation and caching:

1. The client queries SingleStore.
2. SingleStoreDB generates a distributed query plan.
3. SingleStoreDB sends distributed queries to nodes.
4. Individual Nodes generate local plans for the distributed queries assigned to them
5. Query rewrite and parametrize (more below)
6. Hash the query text, version, variables, etc.
7. Check for existing plan
8. Optimize plan
9. Generate byte code
10. Concurrently run and compile the plan
    a. Interpret the query from byte code; and
    b. Compile the byte code to lower level machine language
11. Query Plan is cached
12. Nodes execute local queries.
13. Nodes return data to SingleStore.
14. SingleStoreDB gathers and merges data from the Nodes and returns data to the client

In step 5 “Query rewrite and parameterize”. Wildcards are replaced with explicit table names to remove the need for the interpreter to do this. Values in the WHERE clause are parameterized so similarly shaped queries with different values can reuse this compiled query — making the query significantly more reusable.

In step 11 “Query plan is cached”. The query plan is cached on the node it was compiled and executed on, and is stored both in-memory and on disk. The in-memory plan cache remains for 12 hours since last use, and the on-disk plan cache remains for two weeks since last use. Both plan caches will be invalidated if the administrator runs the ANALYZE command and the statistics for the data are off by a factor of two (double or half), or if the administrator issues the DROP command for the query plan.

Code generation applies to all Data Manipulation Language (DML) queries. In addition, SingleStoreDB generates code during CREATE TABLE and ALTER TABLE statements. These Data Definition Language (DDL) queries generate code in order to reduce the compilation time of subsequent DML queries against the table.

## Separation of Storage and Compute

SingleStoreDB runs with access to a blob store for separated storage. When running with a blob store, SingleStoreDB is different from systems that store all persistent data on the blob store and only transient data on local storage. In S2MS, some data on local disks is the source of truth and must be durably stored. This design allows SingleStoreDB to commit transactions to local disk without the latency penalty of needing to write all the transaction data to blob storage to make it durable. By treating blob storage truly as cold storage, SingleStoreDB is able to support low-latency writes while still getting many of the benefits of separated storage (faster provisioning and scaling, storing datasets bigger than local disk, cheaper historical storage for point in time restores etc.). To understand how S2MS handles data durability in transactions, it's important to understand the role of local durability and blob storage.

Local durability is managed by the cluster on each partition using replication. The in-cluster replication is fast and log pages can be replicated non-sequentially. Replicating non-sequentially allows small transactions to commit without waiting for big transactions, ensuring that commits have low and predictable latency. Data is considered committed when it is replicated in-memory to at least one replica partition for every primary partition involved in a transaction. This means the loss of a single node will never lose data, and if replication is configured across availability zones, loss of an entire availability zone will never lose data. In on-premises implementations, databases commonly commit to local disk but in cloud environments, the loss of a cloud host means almost certain loss of local storage for that host as well.

### Hash Partitioning

Tables are distributed across partitions by hash-partitioning of a user-configurable set of columns called the shard key. This enables fast query execution for point reads and query shapes that do not require moving data between nodes. When join conditions or group-by columns match their referenced tables' shard keys, SingleStoreDB pushes down execution to individual partitions avoiding any data movement. Otherwise, SingleStoreDB redistributes data during query execution, performed as a broadcast or reshuffle operation.

## High Availability

SingleStoreDB maintains high availability (HA) by storing multiple replicas of each partition on different nodes in the cluster. Data is replicated synchronously to the replicas as transactions commit on the primary partitions. Read queries never run on HA replica partitions, they exist only for durability and availability. HA replicas are hot copies of the data on the primary partition such that a replica can pick up the query workload immediately after a failover without needing any warm up.

SingleStoreDB also supports the creation of asynchronous replicas in different regions or data centers for disaster recovery. These cross-region replicas can act as another layer of HA in the event of a full region outage. They are queryable by read queries by default so can also be used to scale out reads.

## Limitless Point-in-Time Recovery

Point-in-Time Recovery (PITR) allows you to recover a database to a transaction-consistent state at any point in time. This means that if a change causes any form of corruption, the database can be returned to a consistent state the moment before the change (or corruption) started. SingleStoreDB ensures all nodes in the cluster are valid and consistent after recovery.

Proper state is established by restoring the most recent snapshot before the intended point-in-time, then playing the logs up to the desired restore point. Additionally, PITR requires no additional software or specialized training and can be restored as many times as needed.

Attaching an unlimited storage database can be faster than restoring an equivalent local storage database. This is because the attach of an unlimited storage database does not copy all data to the cluster, as is the case with the restore of a local storage database. Note that after an unlimited storage database is attached, queries may be slower for some time until remote data is cached locally in the cluster.

A database can be restored to any point in time via the user portal, https://portal.singlestore.com. PITR functionality is available in the Premium and Dedicated editions.

## Conclusion

Modern enterprises require a data platform that is versatile, cost efficient and performant. Not only does the data platform need to support and improve legacy workloads, but also be able to deliver on new business requirements.

SingleStoreDB is the only cloud-native database with the speed and scale to power today's data-intensive applications, SingleStoreDB enables the world's organizations to adapt quickly and embrace diverse, modern data. One platform delivers real-time analytics, eliminates performance bottlenecks and supports massive workloads — driving limitless data experiences like never before.

SingleStoreDB allows for infrastructure convergence, simplicity and support for predictive capabilities in a cost-effective and highly performant manner.