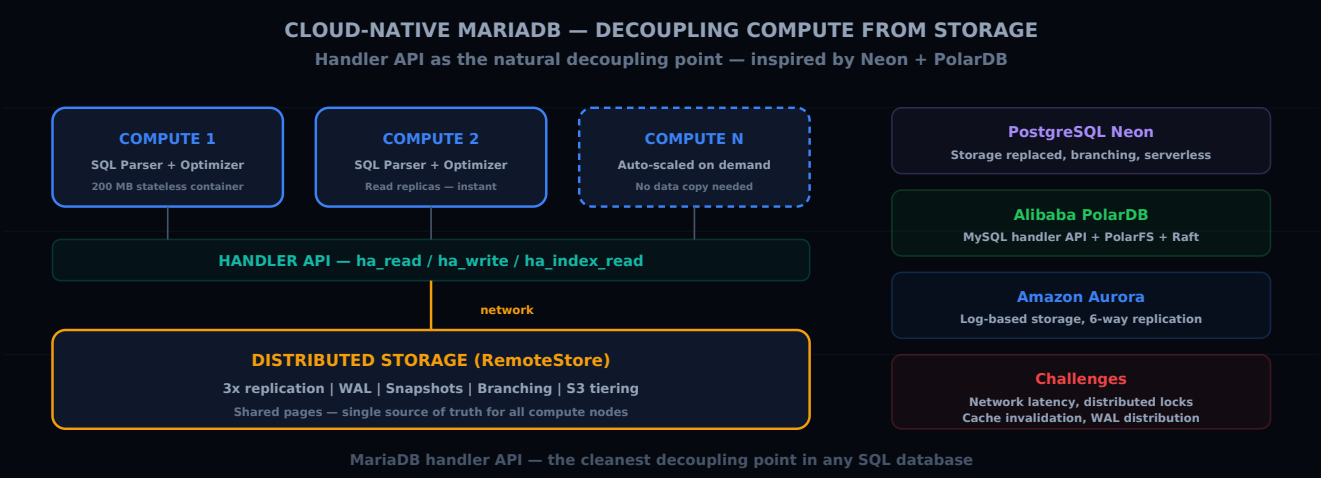
Making MariaDB Cloud Native: Decoupling Compute from Storage

Sylvain ARBAUDIE · July 9, 2025

MARIADB CLOUD-NATIVE ARCHITECTURE STORAGE



The Observation: MariaDB Is Not Cloud-Native

Let us be honest: MariaDB / MySQL in their current form are not cloud-native databases. They were designed to run on a single server with local storage. Even master-slave replication architectures are fundamentally full data copies on each node.

In the cloud, this creates several fundamental problems:

- **Storage is coupled to compute.** If you need more CPU, you must provision a new server with all the storage. If you need more storage, you must resize the server.
- **Horizontal scaling is limited.** Each replica contains a complete copy of the data. Adding a 2 TB slave means copying 2 TB of data.
- **Failover implies downtime.** Promoting a slave involves convergence time during which the most recent writes may be lost (in asynchronous replication) or during which the cluster is unavailable (in synchronous replication).

Modern cloud-native databases like Amazon Aurora, Google AlloyDB, or PostgreSQL Neon have solved these problems by decoupling compute from storage.

The Inspiration: PostgreSQL Neon

PostgreSQL Neon is a fascinating project. It takes PostgreSQL and replaces the local storage layer with remote distributed storage. The PostgreSQL engine functions as a pure compute node: it receives queries, plans execution, and exchanges data pages with a remote storage service over the network.

The advantages are spectacular:

- **Instant scaling**: adding a compute node requires no data copy
- **Branching**: creating a database "branch" (like a Git branch) is instantaneous — it is a metadata operation, not a physical copy
- **Pay-per-use**: inactive compute nodes are stopped, you only pay for storage

Could MariaDB follow the same path?

The Handler API: The Natural Decoupling Point

MariaDB has a unique architectural advantage that PostgreSQL does not: the **handler API**. It is the abstract interface between the SQL engine (parser, optimizer, executor) and the storage engines (InnoDB, Aria, RocksDB, ColumnStore, etc.).

The handler API defines operations such as:

```
handler::ha_open()       // open a table
handler::ha_read_first() // read the first row
handler::ha_read_next()  // read the next row
handler::ha_write_row()  // write a row
handler::ha_update_row() // update a row
handler::ha_delete_row() // delete a row
handler::ha_index_read() // index-based read
```

Each storage engine implements these methods. InnoDB implements them by accessing local B-tree pages. Aria implements them differently. RocksDB uses LSM-trees.

What if a storage engine implemented these methods by accessing **remote** pages over the network?

The PolarDB Approach (Alibaba)

Alibaba has already proven this is possible with PolarDB, which is based on MySQL source code. PolarDB uses:

- A distributed file system (PolarFS) that replaces local storage
- A consensus protocol (Raft) for durability
- A shared page cache between compute nodes

The result is a MySQL whose storage is decoupled from compute. Multiple compute nodes can read the same data simultaneously, and a single node handles writes.

PolarDB shows that the MariaDB/MySQL handler API is a viable decoupling point. Alibaba did not rewrite the SQL engine — they implemented a handler that accesses remote storage.

The Vision: A Remote Handler API for MariaDB

Imagine a MariaDB storage engine called `RemoteStore`:

```
CREATE TABLE users (
    id BIGINT PRIMARY KEY,
    name VARCHAR(255)
) ENGINE=RemoteStore
CONNECTION='storage-cluster.internal:9000/db1';
```

This storage engine would have no local data. Each handler API call would be translated into a network call to a distributed storage service. The storage service would manage:

- Data replication (3 copies minimum)
- Durability (distributed write-ahead log)
- Snapshots and branching
- Compression and tiering (hot data on SSD, cold data on S3)

Compiling MariaDB Without Embedded Engines

The first step toward this vision would be compiling MariaDB without any embedded storage engine. Today, InnoDB is compiled into the `mariadbd` binary. Compilation options allow disabling some engines, but InnoDB remains deeply integrated.

A "headless" MariaDB — a pure SQL engine without storage — would look like:

```
MariaDB SQL Engine (parser + optimizer + executor)
       ↓ handler API
Plugin: RemoteStore → network → Distributed Storage
```

This headless MariaDB would be lightweight (a few hundred MB of RAM), would start in milliseconds, and could be deployed as a stateless container in Kubernetes.

Technical Challenges

This vision is not without challenges:

Network Latency

Every page access goes through the network. Datacenter network latency is on the order of 100 to 500 microseconds. That is 100 to 1,000 times slower than a local SSD access. An aggressive page cache at the compute node level is essential.

Lock Management

InnoDB manages locks locally. With shared remote storage, lock management must be distributed. This is a hard problem that can introduce additional deadlocks and timeouts.

Distributed Buffer Pool

InnoDB's buffer pool is local. In a cloud-native architecture, a cache invalidation mechanism between compute nodes is needed. When one node writes a page, the caches of other nodes must be invalidated.

Transaction Logging

InnoDB's redo log and undo log are local. In a decoupled architecture, the WAL must be distributed and accessible by all nodes.

What Already Exists

Some components of this vision already exist in the MariaDB ecosystem:

- **MariaDB ColumnStore**: a columnar storage engine that can use S3 storage
- **Spider**: a storage engine that distributes data across multiple MariaDB servers

- **CONNECT**: a storage engine that can access external data sources

None of these engines achieve complete compute/storage decoupling, but they demonstrate the flexibility of the handler API.

Conclusion

Making MariaDB cloud-native is an ambitious but not unrealistic challenge. The handler API offers a natural decoupling point that neither PostgreSQL nor Oracle MySQL have in such a clean form.

Alibaba's PolarDB has proven it is technically feasible. PostgreSQL Neon has proven the market demands it. The question is not "is it possible?" but "who will do it first in the MariaDB community?"

The day MariaDB can start as a 200 MB stateless container, connect to distributed storage, and serve queries in milliseconds — that day, the game changes.

This article was originally published on Medium.