



CockroachDB vs Oracle Globally Distributed Database

A comparison of enterprise-grade Distributed SQL offerings.

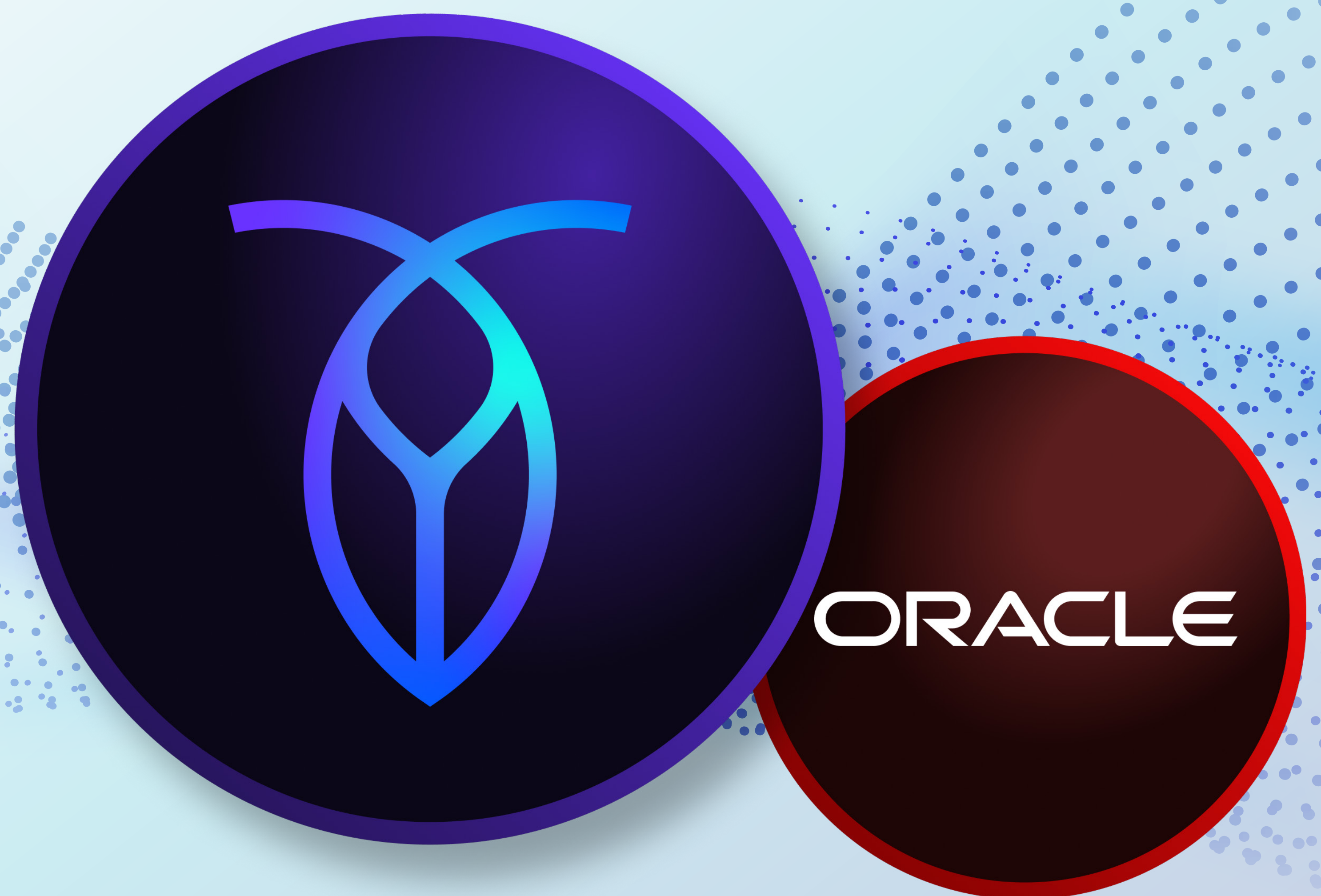


Table of contents

Executive Summary	3
Architectural simplicity	6–7
Feature comparison	8–9
Installation, setup, and operation	11
Feature deep dive	13–16
Conclusion	17

Executive Summary

Mission-critical, data-intensive applications need the reliability and consistency of a relational data store and the flexible scale, high availability, and global coverage of a distributed system. Distributed SQL combines the consistency and structure of the early relational databases with the scalability, reliability, and performance first pioneered in NoSQL. Distributed SQL databases like CockroachDB use this architecture to provide a single logical database that replicates data across physical nodes at scale, on any infrastructure, anywhere in the world.

Oracle Globally Distributed Database (Oracle GDD) claims to surpass leading Distributed SQL databases like CockroachDB by finally capturing the benefits of Distributed SQL that traditional SQL databases struggle to provide: horizontal scale, fault tolerance, and high availability while maintaining data sovereignty and ACID transactions. Brand new to the distributed database market, Oracle GDD is an attempt to copy and match up to — without success — proven solutions like CockroachDB, which have been deployed for years in global enterprise production workloads.

Retrofitted legacy SQL vs cloud native distributed SQL

Released to general availability in March 2024, Oracle GDD partially incorporates the Raft consensus protocol to extend Oracle's legacy partitioning methods across multiple Oracle databases. Oracle itself, however, was built for single-tier applications — not the interconnected, virtualized, and abstracted systems that define cloud computing. Getting this legacy database to perform in an even semi-distributed fashion requires inserting a control plane between the application and multiple physical Oracle database instances ("shards").

Bolting a sharding coordination layer on top is the only way to retrofit a legacy database to handle data distribution and replication functionalities, and to approximate horizontal scalability. This extra layer also means Oracle GDD cannot offer important SQL features like global primary keys, unique constraints, and referential integrity between tables. Oracle recognizes the customer shift to strongly consistent distributed databases built to be cloud native; Oracle GDD is a repackaging of existing software components in an attempt to find relevance in this space. Ironically, this sharding coordination layer is not itself using Raft replication for active-active resilience. This layer, and in particular both the shard director and shard catalog, use legacy high-availability solutions rather than the more resilient Raft protocol.

By comparison, CockroachDB was designed as a distributed relational database from the ground up. CockroachDB is a highly available, ACID compliant, elastically scalable database and has been consistently chosen by enterprises building mission-critical applications. CockroachDB has been in the market since 2015 and has been tested in many different enterprise production deployments, including hyperscale workloads.

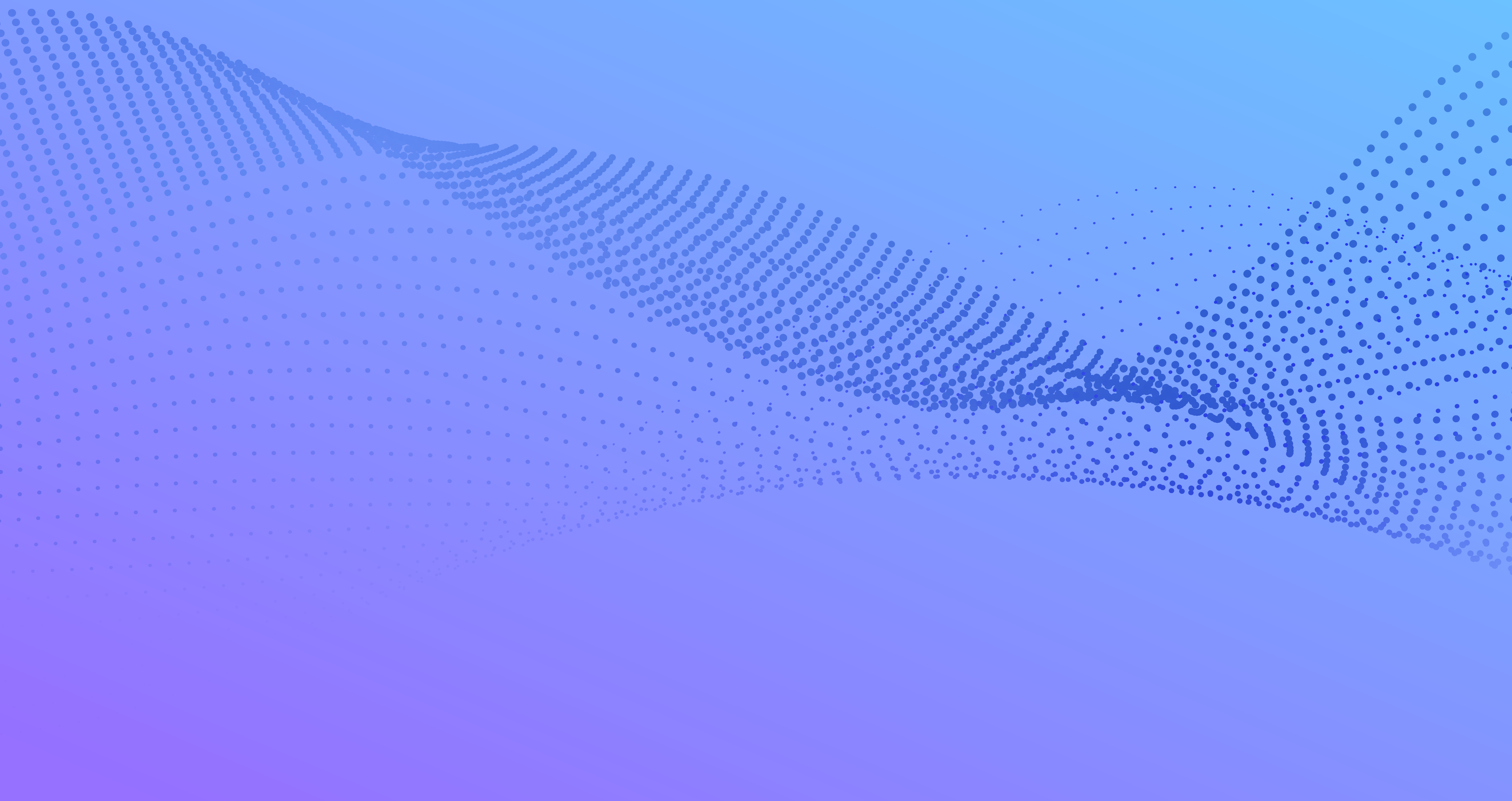
The results speak for themselves. In Performance Under Adversity benchmarking we discovered that CockroachDB was 5x more resilient than Oracle GDD as measured by minutes of outage under adverse conditions. This testing was completed with the free version of Oracle 23ai self-hosted.

Due diligence and disclosures

In this document, we will delve into the practical and technical aspects of both these platforms and highlight why CockroachDB could be the best database of choice for you and your team. It's important to note, however, that, although it's in General Availability, Oracle Globally Distributed Database documentation is still a work in progress. Our team encountered numerous recursive loops, conflicting guidance, and outright gaps in the documentation. This comparative document has been written to the truest and fairest of our ability with the information that is available as of the date of publication (September 2025).

Published September, 2025

Architectural simplicity



Architectural simplicity

A fundamental difference between CockroachDB and Oracle GDD is architectural simplicity vs. complexity, and the resulting stability vs. inherent fragility.

CockroachDB is a distributed SQL database designed for scalability, consistency, and high availability. CockroachDB's distributed SQL architecture allows an application to connect to and execute SQL queries on any node, anywhere in the world, as though it is a single logical database instance. SQL query execution is distributed for performance and every node in a cluster can read and write to the database.

Using the Raft consensus protocol allows CockroachDB to replicate data while surviving failures. Transaction protocols built with distributed data in mind allow CockroachDB to provide guaranteed ACID characteristics without manual sharding or knowledge of shard placement. Since CockroachDB does not rely on explicit data placement to provide ACID transactions, CockroachDB can automatically rebalance replica placement to avoid load hot spots. This is all with no manual intervention, using only a single binary and single running process – fewer services to manage. Schema, replication, and transactions are all managed within the database as part of the database, with no extra configuration required.

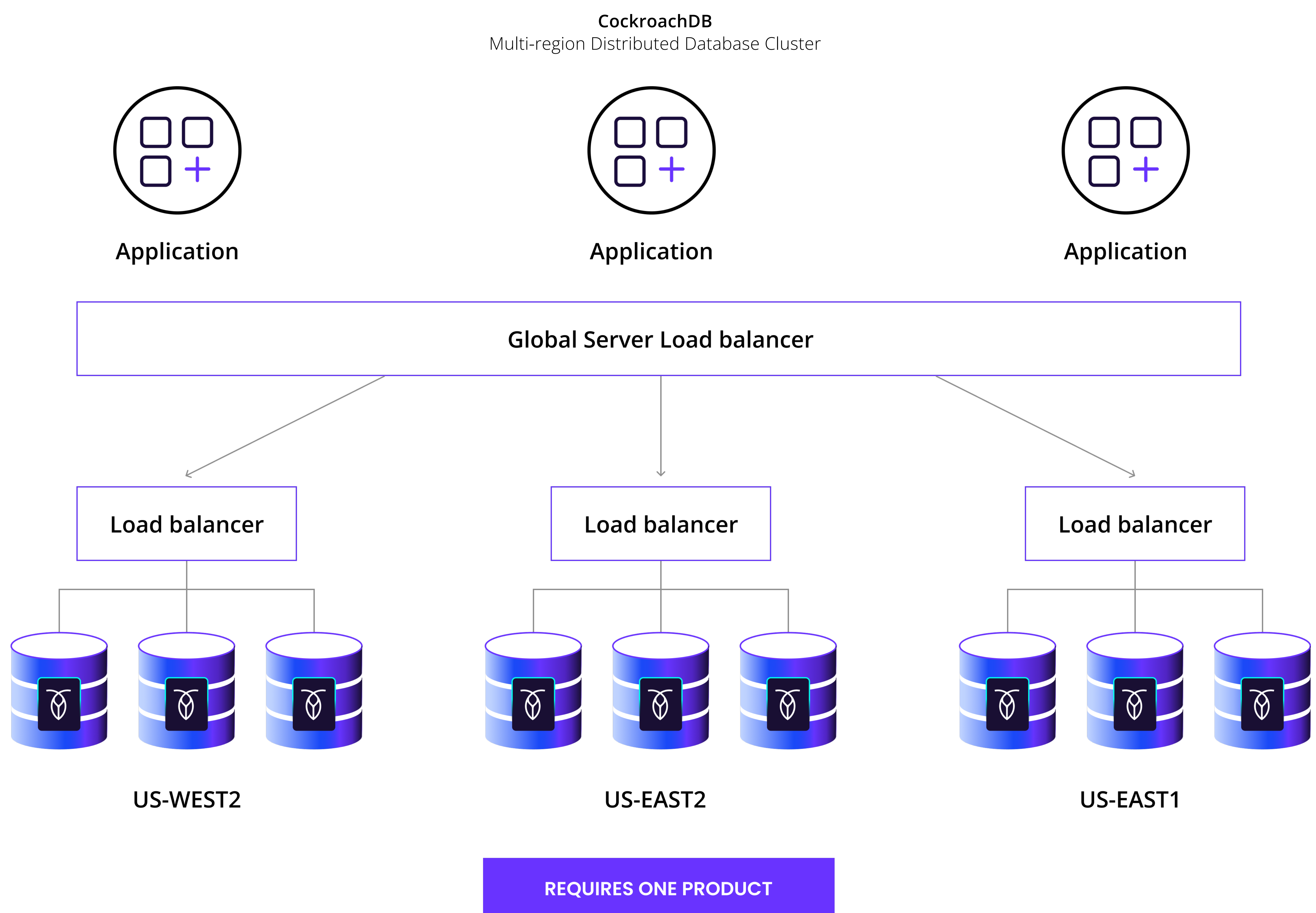


Figure 1: CockroachDB Distributed SQL architecture

In comparison, Oracle GDD works by sharding databases across multiple cloud regions or data centers. Each database is a complete standalone instance with its own storage and compute resources. Data is horizontally partitioned using a sharding key so that each instance holds a unique and separate set of data. Because the databases are unaware of each other, data distribution requires a sharding-aware layer — an extra layer of complexity in between the application and database to add the elements needed to make the architecture feel and act like a distributed database. This extra layer of software does not use Raft for resilience and each component requires its own high availability architecture.

Cross-shard queries are handled by individually querying multiple database instances and stitching the results together. This leads to a lot of limitations, not the least of which are limitations to ACID consistency across shards and the avoidance of data hotspots when managing sharding. The complexity of GDD and its various software components is highlighted in the following architecture diagram.

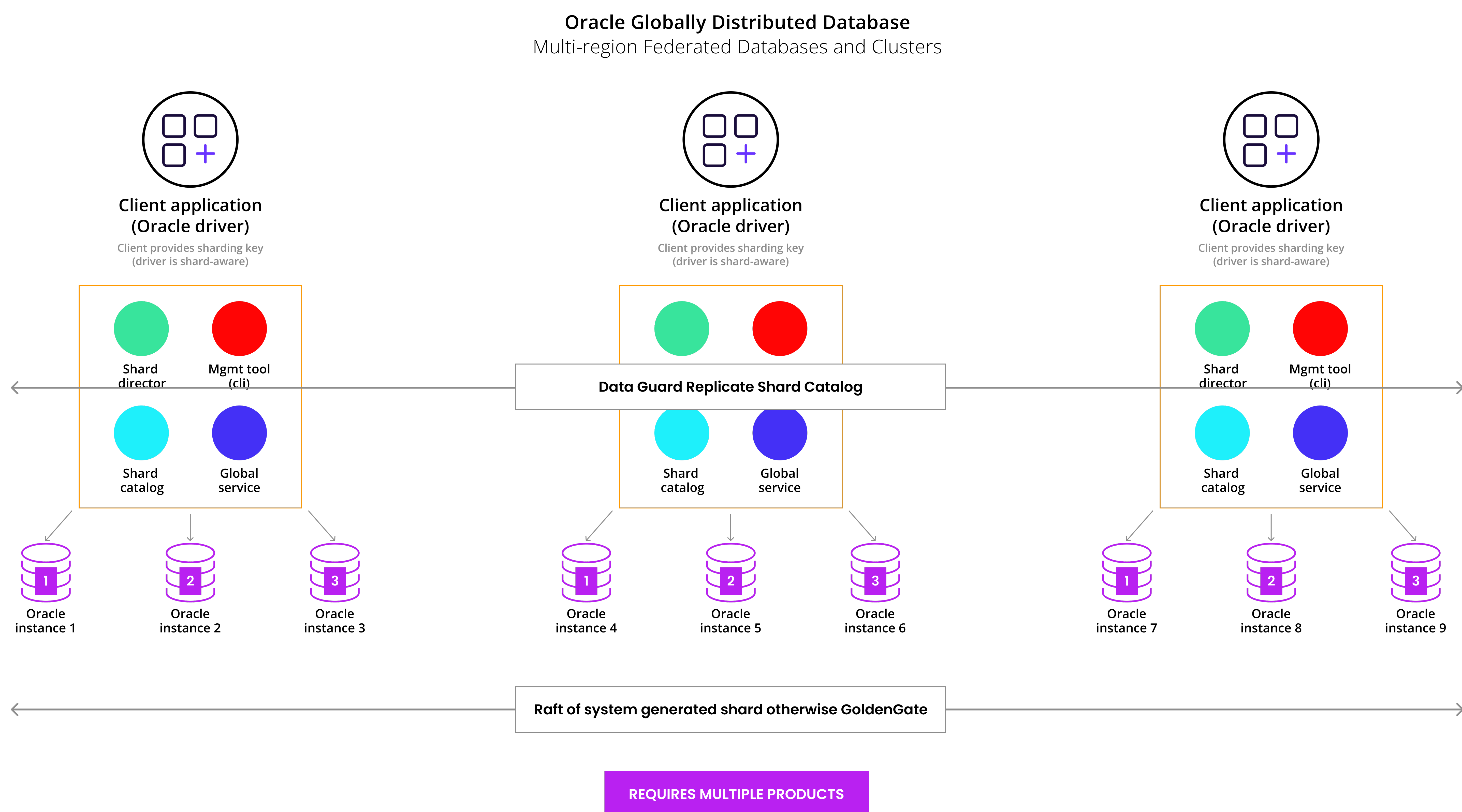
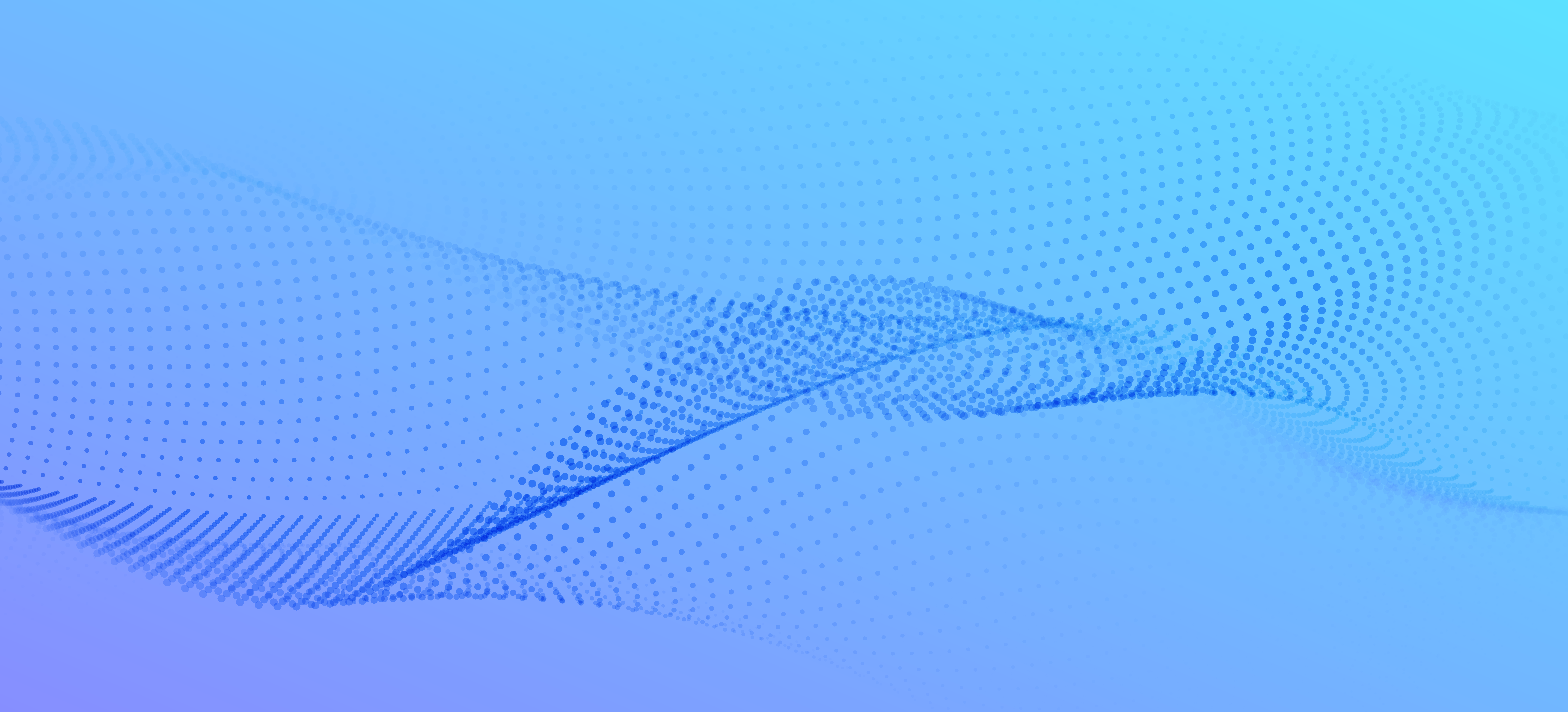


Figure 2: Application architecture using Oracle Globally Distributed Database

CockroachDB's architecture aims for a seamless globally distributed experience through automated sharding, replication, and consensus protocols. Transactions are handled completely within the database, with no need to build an extra logic layer in between the application and the database.

Oracle GDD takes a traditional relational database approach by sharding databases across regions and using a separate non-resilient sharding-aware routing layer to store schema, coordinate and route queries, and handle data consistency and synchronization between shards.

Feature comparison



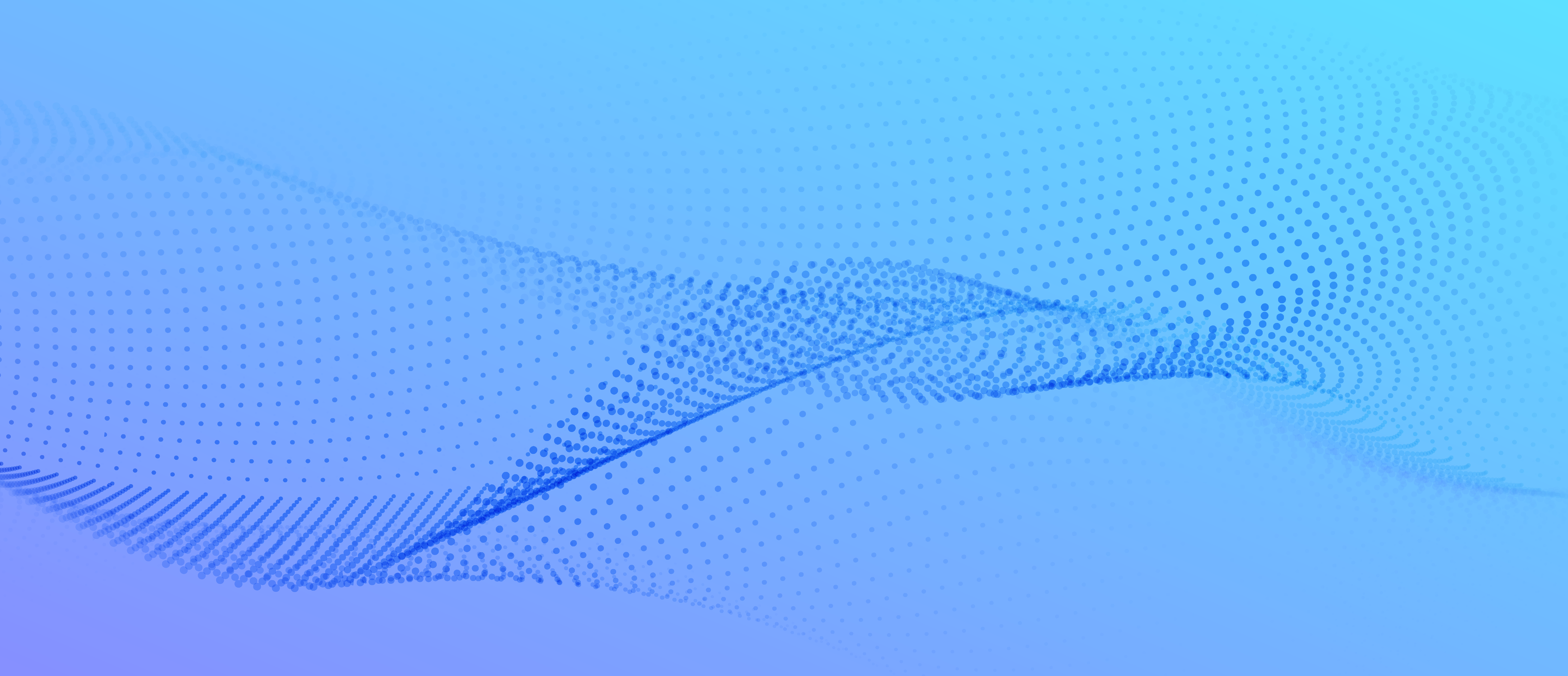
This fundamental difference in database architecture defines core functionality, from availability to scalability, reliability and usability. We will first summarize the core differences between CockroachDB and Oracle GDD in chart form, and then investigate each in more detail and depth.

Feature	CockroachDB	Oracle 23ai	Oracle GDD
Architecture	Shared-nothing distributed database Single logical database cluster One binary to deploy, reducing risk with a simplified application stack Every node can read and write to the database; ACID compliance across the shards/nodes is preserved	Oracle is a single-instance database Uses multitenant architecture where a single container database (CDB) can host many ‘pluggable’ database instances (PDBs). These tenant PDBs may be for one or multiple separate customers all on the same Oracle instance Not capable of performing distributed transactions	A logical representation of multiple independent physical databases. Each physical database is a “shard” Requires separate non-resilient sharding control plane layer for transaction management and query processing whenever >1 shards are used Requires multiple add-on Oracle products (binaries) and increased expertise to achieve a distributed database solution ACID compliance across shards is limited
Installation, setup, and operation	Simple; Single system Multi-active availability built in	Complex, highly manual; must configure and coordinate separate products to achieve high availability; multi-active not available	Complex, highly manual; must configure and coordinate separate products to achieve a mix of high availability and multi-active availability
Deployment capability	Deploy anywhere –across all public clouds, on-prem, hybrid, and bare metal implementations Multi-cloud deployments	Only available on OCI	To get the best experience of GDD, you’ll need to deploy in OCI on Exadata
Scale	Scale horizontally by adding additional nodes Automatic load balancing instantly and intelligently redistributes data between existing and new nodes Every node can read and write to the database	Can achieve high scale via manual sharding or expensive add-ons (Oracle RAC, Golden Gate, etc.)	Achieving high scale requires adding additional physical databases and manually managing sharding Sharding layer is not multi-active, and manually managed with tightly-coupled and expensive add-ons (Oracle RAC, Data Guard, Exadata, etc)
Availability	Multi-active availability with at least three active nodes to distribute data and perform writes. Database can scale beyond single machines because all nodes in a cluster serve reads and writes Built-in high availability and fault tolerance; set availability goals at the database, table, or row level High availability on single cloud, multi-cloud, self-hosted, or hybrid deployments	High availability best achieved using expensive add-ons such as Oracle RAC, many of which only run in OCI. Requires configuring and maintaining a dedicated “Fast Recovery Area” on a separate disk group	Oracle GDD offers multi-active replication for shard replicas but not for the sharding layer Sharding layer is only available as high-availability, and then only with expensive add-ons
Data distribution and replication	Native to CockroachDB automatically replicates, distributes, and rebalances data: automatically avoids data hotspots	Not available	Raft replication reconfigures replication settings across shards Replication factor limited to 3
Data locality	Geo-locates and ties data to a locality for latency reduction and optimal query performance Data locality can be defined at the database, table or even row level for compliance with data regulations	Simulates geo-partitioning via “native database sharding,” but additional manual operations are needed to ensure data remains globally consistent and locally performant	Geo-partitioning requires that the shard key is (or involves) the locality Must have a physical database (shard) in each region where you need to geolocate specific data Data locality defined at database level only, manual management required

Feature	CockroachDB	Oracle 23ai	Oracle GDD
Data loss	3x data replication by default with concurrent writes across replicas	Does not offer a single no-data-loss, multi-region, active-active solution “Snapshot” serializability and defaults to read-committed availability; multi-active not available	Does not offer a single no-data-loss, multi-region, active-active solution “Snapshot” serializability and defaults to read-committed
Uptime SLA	99.999% uptime	Varies according to Oracle license and add-on services	Oracle GDD SLA: 99.95% uptime (with Oracle Globally Distributed Autonomous Database serverless implementation on OCI)
RPO	RPO=0 for data and metadata	On OCI, approximately five minutes if using point-in-time recovery feature; RPO of 24 hours if using standard daily backup	RPO=0 for data only
Awareness	Database nodes are natively distributed and locality aware	Single instance database; awareness is not an issue	Database shards are fully isolated and unaware of each other Data distribution and locality logic is reliant on separate shard management layer outside of database
Reads and writes	Cross-node SQL queries handled internally and automatically Advanced commit protocol enables global indexes Cross-regional transactions are optimized to reduce latency; transactions include all secondary indexes Each node can service both reads and writes	No support for global indexes. Only allows local indexes within a shard	Manually defined sharding keys within routing layer distribute each query to the shard where that data is located No support for global indexes. Only allows local indexes within a shard No distributed query execution
ACID transactions	Guaranteed Serializable isolation by default Offers read committed	Supports non-distributed ACID transactions not 100% autonomous	ACID transactions not available when data required for that transaction is split onto on multiple shards/instances
Database maintenance and schema changes	Zero downtime maintenance is built in CockroachDB allows online schema changes and rolling upgrades	Downtime is required for updates Availability can be improved by adding on additional Oracle services at additional cost (RAC, Exadata, Data Guard, Oracle Fleet Patching & Provisioning, etc) Schema changes are theoretically possible with highly manual application multi-versioning*	Downtime is required for updates Availability can be improved by adding on additional Oracle services at additional cost (RAC, Exadata, Data Guard, Oracle Fleet Patching & Provisioning,etc) While schema changes appear to be online, the operations still require user intervention and are not 100% autonomous

*For example, version 1 of your app supports schema A, and you want to introduce schema B. First you release app version 2 that is able to work with both schemas A and B. Then with version 3 you update the schema to B. Last you deprecate support for schema A in version 4. This is one of the reasons NoSQL was invented.

Installation, setup, and operation



Installation, setup, and operation

CockroachDB is designed to be an all-in-one database that is simple to set up and operate. It is a single distributed database binary with all the components for high availability, replication, and scaling built in. It can be deployed anywhere: on-premise, within containerized environments such as Kubernetes, and on the three major cloud providers for single- or multi-cloud deployments, as well as hybrid deployments.

The key steps to setup CockroachDB are:

- 1 Download the CockroachDB binary
- 2 Start the initial nodes
- 3 Add more nodes to the cluster as needed, up to 300 nodes per cluster are supported

CockroachDB automatically handles intra-node replication, cross-region replication, and data distribution across nodes without any additional tools or licenses required. This simplifies deployment and operations, and avoids hotspots.

In contrast, the key steps setting up Oracle GDD are numerous and highly manual. Deploying Oracle GDD involves integrating and configuring multiple separate enterprise products and tools.

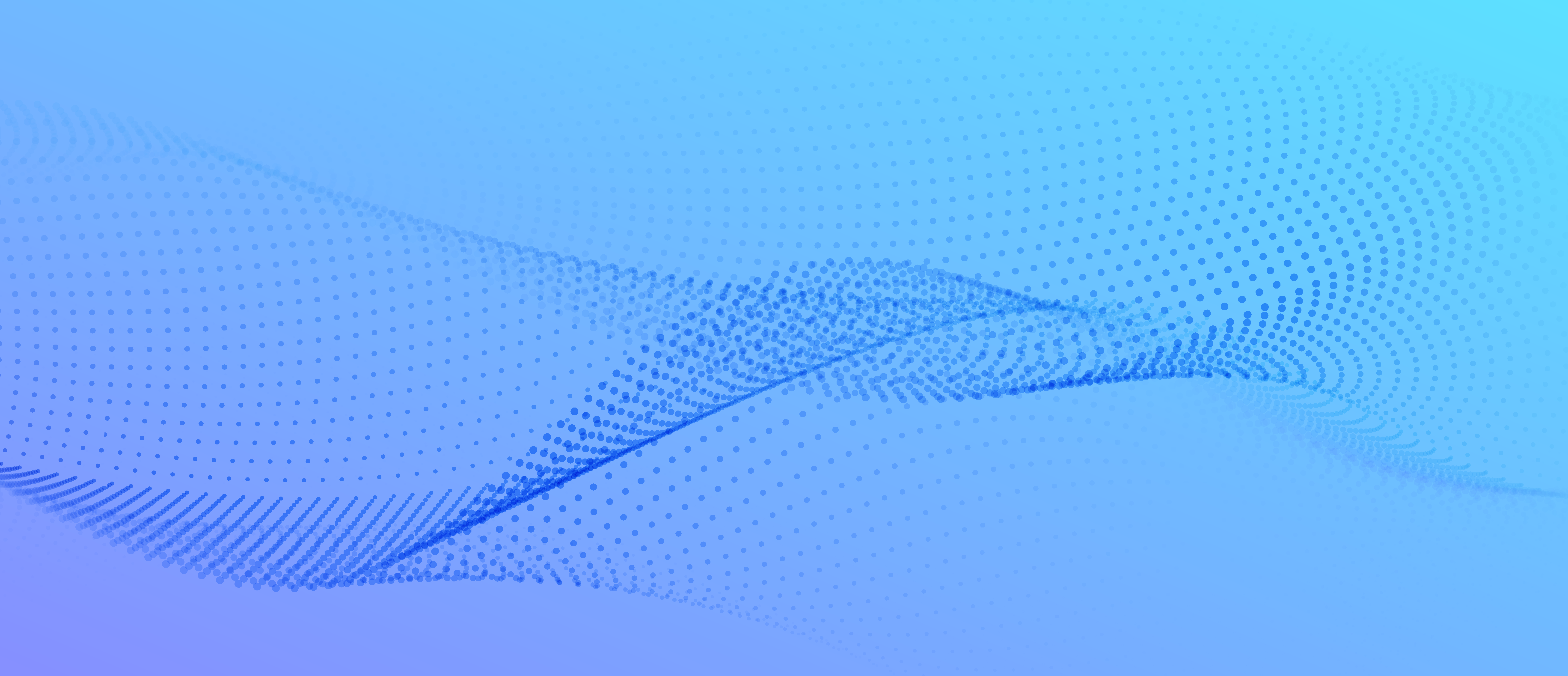
The key steps to setup Oracle GDD are:

- 1 Setup multiple Oracle 23ai databases
- 2 Configure Data Guard for HA between Shard Catalog and Shard Director
- 3 Set up Global Data Services for inter-site routing
- 4 Use Raft for controlling data distribution
- 5 Configure RMAN for backup and disaster recovery

Many of these components, like RAC and Global Data Services, require separate paid enterprise licenses from Oracle. The complex architecture with many moving parts makes Oracle's initial setup more complicated, time-consuming, and costly compared to CockroachDB's integrated and straightforward setup and configuration.

This complexity makes deploying Oracle GDD expensive, even for users under an Oracle ELA, because significant resources have to be dedicated to simply operating, scaling, and maintaining the database. Since Oracle GDD's separate sharded database instances are fully isolated and shared-nothing, each instance must be individually maintained, patched, updated, etc. Horizontally scaling Oracle GDD requires significant hands-on developer intervention to add a new shard, then individually update the others shard while monitoring the end to end process across all the components. Licensing costs aside, Oracle GDD comes with hidden operational costs that scale up as businesses scale — costs that CockroachDB can reduce and, in many cases, eliminate entirely.

Deep dive: features



Data distribution and replication

Another key difference between CockroachDB and Oracle GDD lies in how each approaches data replication.

CockroachDB is designed as multi-active replication with multi-node writes, which means that all nodes in the cluster can handle reads and writes for higher availability. This makes CockroachDB ideal for application architectures designed for disaster prevention and operational resilience.

Oracle GDD, on the other hand, uses an active-passive approach for replication across sharded instances at different data centers. One sharded instance is designated as the primary or “active” site, while other data centers act as “passive” standbys. Writes can only occur at the primary site, with changes asynchronously replicated to the passive standby sites. Oracle GDD thus provides disaster recovery capabilities but does not offer the same disaster-prevention level of availability as CockroachDB’s multi-active model during failures.

Another primary benefit from CockroachDB’s approach to data replication is that the database can automatically redistribute replicas to balance load. This built-in feature enables efficient use of hardware and avoids single-node bottlenecks. Oracle GDD’s data replication approach, however, requires stringent, manual time consuming pre-planning for how to handle sharding and shard redistribution strategy.

Scale

CockroachDB allows for scaling of reads and writes simply by adding nodes as required. This means that as workload volume grows, CockroachDB can continue to handle increased read load by automatically splitting and redistributing ranges without any need for manual intervention or sharding. This significantly reduces operational complexity and overhead, allowing your team to focus on other important tasks.

When a database is written to be horizontally scalable, you effectively remove the scale ceiling associated with relational databases. When you can add additional compute and storage to your DB system by adding additional commodity nodes, it removes the need for expensive and exotic hardware solutions like RAC and Exadata.

Oracle GDD, however, works by [horizontally partitioning data across shards](#) based on sharding 10 keys defined by the application or DBA. New shards can be added to scale the database as workloads increase, but it is a labor-intensive operation. First, manual intervention is required to create the new database shard, followed by manual operational toil to repartition historical data and update sharding keys across all instances. Also, as more shards are created, the amount of cross-shard SQL increases considerably. This creates more overhead and can begin to impact query latency and performance, as well as manageability. At the same time Oracle’s replication factor is limited to three, which limits the amount of resilience available to data consumers.

Distributed ACID transactions

CockroachDB provides strong consistency across all nodes in the cluster, with 3x data replication by default with concurrent writes. Every transaction occurs with serializable isolation and with ACID guarantees.

Oracle GDD, on the other hand, can guarantee ACID transactions *only when the entire process occurs within a single shard*. This means Oracle GDD does not offer a single no-data-loss, multi-region active-active solution, and can offer only “snapshot” isolation with default to read-committed.

[Oracle's own documentation](#)¹ states that multi-shard transactions in Oracle GDD are not atomic, and therefore not consistent, since the database iterates over each shard: To run this UPDATE statement [query] on all shards, the shard coordinator iterates over all of the primary shard databases and invokes the UPDATE statement remotely. The coordinator starts a distributed transaction and performs a two-phase commit to guarantee the consistency of the distributed transaction. If there is an in-doubt transaction, you must recover it manually.

Thus, if you have a multi-region deployment with Oracle GDD, potential consistency issues are possible — particularly for the split-second transactions in financial, gambling, and other sectors where data consistency is crucial. CockroachDB is designed to provide a truly consistent experience all of the time, with strong consistency even across multiple regions and clouds. All CockroachDB reads and writes will be consistent across all nodes in the cluster, all of the time, regardless of where they are located. We cover this extensively in this [video](#)².

Maintenance and schema changes

Schema changes are an ever-evolving requirement for application teams who are focused on creating the best user experience. To enable this experience in a fast and agile environment, CockroachDB supports online schema changes by default. This means that you can change the schema of your database without having to take the database offline, and schema changes propagate seamlessly across nodes with zero operational disruption. [Here is a video](#)³ demonstrating how online schema changes can be done with zero downtime on CockroachDB.

Oracle GDD, on the other hand, does not support schema changes without some level of downtime (although it is possible to somewhat reduce planned downtime by purchasing additional Oracle products). Oracle GDD uses distributed transactions and two-phase commits to support limited DML statements; many common DML operations (inserts into multiple tables, merge statements involving multiple shards) are not supported at all.

[Schema changes in Oracle GDD](#)⁴ are complex and cumbersome: Sharding keys must be meticulously specified and must be fields that rarely change. To implement a schema change in Oracle GDD, you first need to set a session variable that indicates that you're doing a cross-shard operation. Sharding keys must be included in all tables that are stored together; joins to tables stored together must include the sharding key. Primary keys must include the sharding column. Finally, using Oracle GDD's external CLI tool, you must check each individual shard to verify that the operation happened successfully. This entire process must be repeated for each individual database instance, each time requiring you to take a portion of the database offline to change the schema, resulting in downtime.

¹ docs.oracle.com/en/database/oracle/oracle-database/23/shard/query-and-dml-processing1.html#GUID-E3CF0022-FA38-4DD0-A5D5-7A409EA45F76

² "The Importance of Strong Consistency." www.youtube.com/watch?v=a5N9ot-9Flc

³ "How to perform zero downtime schema changes." www.youtube.com/watch?v=xvBBQVIGYio

⁴ docs.oracle.com/en/database/oracle/oracle-database/23/shard/schema-design1.html

Upgrades, maintenance, and repaves

CockroachDB can be upgraded one node at a time, with zero impact on your application. This capability comes from the database's quorum replication: even if a single node is down, there are sufficient copies to serve the queries. Each node in a CockroachDB cluster is self-contained and can operate independently, but since each node is also aware of the data on the rest of the cluster it can serve queries based on that overall knowledge.

When a node is upgraded, it is seamlessly replaced by a new node running the latest version of the software. This process is transparent to your application, so you can continue to use the database without interruption. This also applies to repavings (rebuilding servers) due to software vulnerability updates or OS updates, etc. As data is replicated multiple times in CockroachDB, we can ensure that a quorum (majority of the data replicas) is always available. Thus, even when we repave the node, you can expect zero downtime.

Oracle GDD does not offer the same level of flexibility and scalability for rolling upgrades nor does it offer rolling repaves by default. Upgrading the Oracle Globally Distributed Database environment, however, requires upgrading in a very specific sequence such that the shard catalog is upgraded first, followed by the shard directors, and finally the shards themselves. Executing this complex process requires a substantial amount of manual overhead every time you need to update or patch the database.

Node failure and self-healing

CockroachDB can automatically recover from node failures with zero intervention from database operators. This self-healing capability automatically triggers when a node fails; CockroachDB will automatically up-replicate data of a failed node onto the remaining ones, in order to restore the desired replication factor. This returns the cluster to a healthy state, ready to weather another disruption⁵.

By contrast, there are multiple failure scenarios possible in Oracle GDD nodes. What if the shard coordination layer loses access to the shard nodes, or the shard nodes lose access to other nodes? How do shards in one region act when in a network partition, or in the event of network or storage jitter? These are all relatively common scenarios and, in Oracle GDD, you will need a different plan to recover the database in each type of event. Application teams must also have a reconciliation process to confirm that data was asynchronously copied from one region to another. This is a time-consuming and error-prone process. Users of CockroachDB do not need to have such planning contingencies in place because of CockroachDB's self-healing capabilities.

⁵ www.cockroachlabs.com/docs/stable/demo-fault-tolerance-and-recovery

Replication factor

The number of failures that can be tolerated is equal to $(\text{Replication factor} - 1)/2$. For example, with 3x replication, one failure can be tolerated; with 5x replication, two failures, and so on.

Oracle is limited to a replication factor of 3⁶, where CockroachDB doesn't have such a limitation. This means that Oracle can't survive two simultaneous failures as it would be able if it supported a replication factor of 5.

A replication factor of 5 is important for a very common real-world use case - that of regular maintenance. Let's say you upgrade linux on the server. It requires the server to be taken out of the cluster for perhaps tens of minutes, or even half-an-hour. Doing this on all nodes in a cluster can take an entire day. During that upgrade, you have two choices: either allow the cluster to up-replicate, which requires a LOT of rebalancing, perhaps affecting the workload, or you take a risk and hope that while the cluster is under maintenance nothing else fails. A database with a replication factor of 3 can only survive one failure - and that "failure" is being used by the upgrade. A second failure would bring the application down. Running with a replication factor of 5 is a cost effective way of ensuring that even during regularly scheduled online maintenance your cluster can survive an additional node failure.

Deploy anywhere, highly available all the time

Although Oracle GDD is compatible with hybrid-cloud deployments, and is available to run on other cloud platforms, it is designed to work best on Oracle Cloud Infrastructure. More importantly, Oracle GDD requires add-on Oracle products and services (including Exadata) to achieve the high fault tolerance and high availability that are native to Distributed SQL — and these Oracle products are available only when an application is deployed to Oracle's own cloud infrastructure. To gain support for Oracle's Raft-enabled version of GDD (Oracle 23ai), GDD must be running on OCI. And to get the best experience of GDD, you'll need to deploy in OCI on Exadata.

CockroachDB, on the other hand, is fully cloud agnostic. It can be deployed across any of the major cloud providers (or several/ all of them, for a multicloud approach). It also offers hybrid cloud and on-premises deployments, making CockroachDB compatible for organizations that want to move to the cloud gradually, to repatriate data from the cloud, or who prefer to deploy infrastructure in a hybrid environment.

⁶ <https://docs.oracle.com/en/database/oracle/oracle-database/23/shard/raft-replication-concepts.html#SHARD-GUID-AF14C34B-4F55-4528-8B28-5073A3BFD2BE>

Conclusion

Choosing the right technology for the right use cases can often feel daunting. Oracle GDD has a number of limitations as described in this document, which may mean it is not the best database choice for every application or use case. Here are a few examples of enterprises that have chosen CockroachDB over Oracle GDD in production that can help you with a user perspective as you begin this journey.

- ✓ Financial data firm consolidates multiple Oracle databases into a single global CockroachDB deployment
- ✓ How and why two banking giants migrated from Oracle to CockroachDB
- ✓ How Form3 is building unkillable, global payments infrastructure

Overall, CockroachDB is a powerful and versatile database. Connect with us to learn more about how you can build your applications to be robust, consistent, and scalable with CockroachDB.

About Cockroach Labs

Cockroach Labs is the creator of CockroachDB, the cloud-native, distributed SQL database designed to enable global businesses to build and operate world-changing applications. CockroachDB is trusted to run mission-critical workloads for some of the world's largest enterprises, including Bose, Comcast, HardRock Digital, Fortune 50 financial institutions, as well as category leaders in retail and media.

To learn more, visit www.cockroachlabs.com

