CrateDB

ARCHITECTURE GUIDE

CrateDB Key Concepts

January, 2024

CrateDB

Table of Contents

Architecture Overview

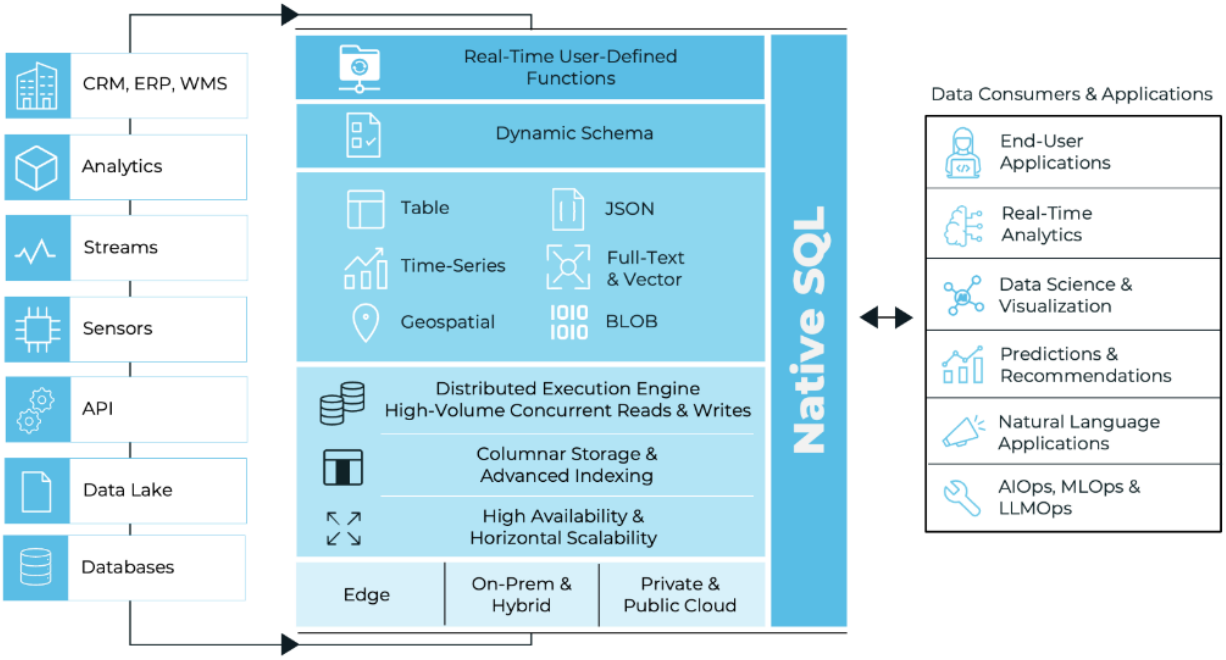
CrateDB allows you to store any kind of **structured**, **semi-structured** and **unstructured** data in one single technology. It also offers a **dynamic schema** for rapid evolution and provides access to data via **SQL**. It streamlines data management by blending operational and analytical data to enable modern application development. It offers the best of SQL, NoSQL, and search engines and enables rich full-text and vector similarity search to power both traditional and AI-enabled applications, ranging from real-time analytics over machine learning, to natural language processing and LLMs.

Its distributed, shared-nothing architecture enables **high availability**, **vertical and horizontal scaling** as well as **high-volume concurrent reads and writes**.

CrateDB does not just provide faster time-to-market, but also very low operational overhead. **Scaling** is a matter of adding nodes, the database itself takes care of data distribution.

The combination of advanced indexing and columnar storage enables **very fast** queries in single-digit milliseconds across billions of rows.

The **flexible deployment options** provide the level of flexibility needed for any use case: as a fully managed service on AWS, Azure and GCP; as a self-deployed solution in your own cloud environment or on-premises; as a solution deployed at the Edge.



CrateDB Architecture Overview

Open Source Licensing Model

CrateDB embraces and supports the open source development model. Its source code is available on GitHub under the Apache 2.0 License, allowing users to access and contribute to its ongoing development.

On one hand, CrateDB offers the flexibility of self-deployment, enabling organizations to customize and tailor the database to their specific needs.

On the other hand, users can access CrateDB as a service through the CrateDB Cloud offering, simplifying the setup and management process.

For all deployment models, support subscriptions are available on demand, with “Basic” and “Premium” options.

CrateDB Cloud paid plans include “Basic” support by default.

Multi-Model Database

CrateDB is a **multi-model database**. Its strength lies in the **efficient handling of multiple data models within the same database and even within the same table**:

- structured data (tables, time series, geospatial)
- semi-structured data (JSON / documents)
- unstructured data (text, vector, binaries like documents, images, and videos)

It eliminates the need to manage and synchronize multiple database technologies and learn different languages by offering unified access via the well-known SQL language.

- ☑ All data models can be combined in the same record.
- ☑ All data models are **accessible via SQL**, the well-known query language, allowing for complex queries, full-text and vector search.
- ☑ Complex objects and nested objects can be stored with no human intervention. Data can be directly inserted as a JSON string.
- ☑ New columns, supporting any data type and format, can be dynamically added, without table locks, allowing for seamless adaptation to changing needs and requirements. An important CrateDB's strength lies in its schema flexibility.
- ☑ CrateDB addresses the traditional challenge of the rigidity inherent in relational schemas and the complexity associated with making changes in a production environment.
- ☑ Columnar storage enables fast aggregation across individual records, making it suitable for combined operational and analytical use cases.
- ☑ Binary large objects (BLOBs) can be stored separately from the main database workload, reducing storage costs, with streamlined read/write access via the HTTP REST API.

CrateDB can work with data from various sources – enterprise application data (like CRM, ERP), analytics data, streams data, sensor data, APIs, and data from data lakes or other databases – and can accept many formats with no need for complex data transformations.

Document / JSON

CrateDB supports the storage of JSON-based documents into OBJECT typed columns, offering flexibility for complex multi-model data structures with diverse attributes, nesting levels, and arrays of objects.

- CrateDB simplifies SQL access to nested documents by enabling direct and nested attribute querying. This feature allows users to navigate and retrieve data seamlessly within JSON structures using native SQL syntax.
- Inserting documents and JSON payloads is straightforward through JSON strings, which are automatically parsed and cast into the defined datatypes.
- Alternatively, dedicated object literals, closely resembling JSON syntax, provide full control over used datatypes, especially when dynamically creating new attributes.
- Different policies exist to enforce schemas, work schema-free, or follow a hybrid approach that enforces certain attributes and datatypes. Three options – 'strict', 'dynamic', or 'ignored' – are available for defining the level of flexibility allowed for both objects and new columns in the schema.
- Automatic indexing and columnar storage are applied to any attribute in the JSON-based documents, depending on the column policy.

Relational Data

No matter the data stored in CrateDB, it is represented in tables and columns, albeit in a different manner in the background. This simplicity facilitates a quick start with CrateDB, allowing the use of its native SQL for reading and writing data.

In comparison to traditional relational databases, CrateDB offers:

- Higher concurrent read and write rates
- Support for high data versatility
- Real-time query performance
- Horizontal scalability

While CrateDB does not support ACID transactions, atomic operations and durability is guaranteed at the record level. Queries by primary key always return the latest results. The choice of eventual consistency prioritizes high-availability and partition tolerance, ensuring resilience to most hardware and network failures. In conjunction with Multi Version Concurrency Control, the storage engine can handle high-volume concurrent reads and writes.

Full-Text Search & Search Engine

Rich search capabilities are essential for almost any modern application, with the user experience set high by powerful and user-friendly search engines like Google or Bing.

- CrateDB meets the demand for quick and accurate search in textual data, leveraging the robust full-text search capabilities of Apache Lucene.
- Dedicated full-text indexes are updated in real time using powerful analyzers supporting more than 30 languages, enabling instant queries across billions of records with no latency.
- CrateDB ensures scalability through its shared-nothing architecture and horizontal scaling capabilities.
- The SQL interface exposes most of Lucene's powerful query syntax, allowing users to effortlessly perform complex search queries, including boolean logic, wildcard searches, phrase searches, proximity searches, fuzzy search capabilities, and synonyms.

Vector Store & Similarity Search

CrateDB's integrated vector data type and similarity search capabilities streamline data management by eliminating the need for an additional vector database. This reduction in complexity cuts down development time and the total cost of ownership. It ensures alignment between your (meta-)data and vector representations without the need for intricate data synchronization processes.

- Any table can be enhanced with a (nested) column of type FLOAT_VECTOR that utilizes an HNSW algorithm for efficient indexing, with a maximum of 2048 dimensions.
- The built-in approximate k-Nearest-Neighbour Search (kNN) retrieves similar vectors based on the Euclidean distance. In combination with lexicographical full-text search, this approach significantly increases search precision and relevance.

The vector store and kNN search serve as the gateway to Generative AI, allowing users to efficiently store embeddings generated by their preferred machine learning models. The similarity search offers relevant context for prompts in Large Language Models, facilitating Retrieval Augmented Generation (RAG) pipelines. Thanks to CrateDB's integration with LangChain and native SQL support, users can rapidly build and implement such solutions.

Time Series Data

CrateDB **is very well suited for time series data:**

- **High Cardinality:** robust sharding and partitioning capabilities enable time-based partitioning, facilitating the long-term storage of time series data without the need for aggregation and downsampling. Preserving granular details is crucial, providing enhanced flexibility for revisiting historical data, gaining new insights, and conducting precise forecasting, essential for strategic decision-making processes.

- **Data Tiering:** With CrateDB, you can efficiently move older partitions to slower but cost-effective spinning disks, while keeping recent data on fast SSDs. This allows for fast query speed on the most recent data without losing details in older data.

- **Broad Data Model Support:** CrateDB accommodates various data types that can be combined with time series data. Enrich your time series data with additional context from documents, track fleets with geospatial information, calculate vectors/embeddings, and join across different time series. This versatility eliminates the need for new costly technologies with complex maintenance and data synchronization.

- **Distributed Architecture:** CrateDB's distributed storage seamlessly scales horizontally across multiple nodes using commodity hardware. This makes it ideal for time series workloads involving large volumes of data from diverse sources (sensors, IoT gateways, CRM, ERP) that require real-time ingestion, enrichment, and processing to serve multiple simultaneous data consumers who demand instant access to information.

- **Rich SQL:** CrateDB includes built-in time-series functionality, such as window functions and time-based indexes, facilitating easier querying and analysis for high performance. The JOIN operator enables the combination of time series data in one table with corresponding metadata in another table, preventing the need to push excessive data into your application.

- **Columnar Storage:** In addition to efficiently handling many columns, CrateDB utilizes a columnar storage format to ensure efficient aggregations over large volumes of data.

Spatial Data

Spatial data often generates substantial volumes of information. CrateDB provides scalable SQL support for geospatial data types and functions tailored for applications such as fleet tracking, mapping, and location analytics.

- In many use cases, including data analysis and machine learning, location data plays a crucial role. CrateDB accommodates this by storing and querying geographical information using the **geo_point** and **geo_shape** data types.
- Users can fine-tune geographic index precision and resolution to achieve faster query results. Additionally, exact queries can be performed using scalar functions like **intersects**, **within**, and **distance**.

Multiple geospatial datatypes are supported:
`Point`, `MultiPoint`, `LineString`, `MultiLineString`, `Polygon`, `MultiPolygon`, and `GeometryCollection`.

BLOB Data

CrateDB provides robust support for storing binary large objects (BLOBs) with advanced cluster features for efficient replication and sharding of these files. By treating BLOBs like regular data, CrateDB ensures their optimal distribution across multiple nodes in the cluster.

To integrate BLOBs into CrateDB, creating a dedicated BLOB table is necessary. This table can be sharded to distribute binaries effectively across nodes, promoting scalability and improved performance. Users have the flexibility to define a custom directory path exclusively for storing BLOB data. This distinct path can differ from the regular data path, allowing for the segregation of normal data stored on fast SSDs and BLOB data stored on large, cost-effective spinning disks. The storage path can be globally set or specified during the creation of a BLOB table.

Interacting with BLOB tables in CrateDB is made easy through the use of the HTTP(S) protocol. For uploading a BLOB, the SHA1 hash of the BLOB serves as its unique identifier. To download a BLOB, a simple GET request to the appropriate endpoint is sufficient, and for deletion, a DELETE request can be sent.

Native SQL Syntax

CrateDB is designed with scalability and ease of use at its core, emphasizing support for SQL to seamlessly work with data and integrate into a wide ecosystem.

CrateDB offers a standards-based SQL implementation with a particular focus on achieving ANSI SQL compatibility. While maintaining these standards, CrateDB's SQL dialect introduces unique characteristics to fully support the multi-model approach.

- The rich SQL dialect in CrateDB covers a **broad spectrum** of operations, including joins, aggregations, indexes, subqueries, common table expressions, user-defined functions, and views.
- Additionally, it includes support for **full-text search**, **similarity search, geospatial queries**, and **nested JSON object columns,** all accessible through SQL.

Furthermore, CrateDB implements the **PostgreSQL Wire Protocol**, ensuring compatibility with most SQL tools using the PostgreSQL drivers.

- It provides two primary client interfaces: one via the PostgreSQL Wire Protocol and another through an HTTP endpoint.
- CrateDB is designed for access by SQL-enabled ETL, reporting, and development frameworks, offering users the flexibility to integrate with a wide range of modern data stack tools. This approach ensures that users are not restricted to tools specifically designed for certain NoSQL engines, thereby avoiding vendor lock-in and promoting a versatile and adaptable data management environment.

CrateDB implements most of the familiar SQL syntax. For details on the specific syntax and please refer to the **latest reference documentation** to learn more about the specific syntax and **differences to ANSI and PostgreSQL.**

For a detailed list of supported operations, please to the information_schema.sql_features table.

Distributed Shared-Nothing Architecture

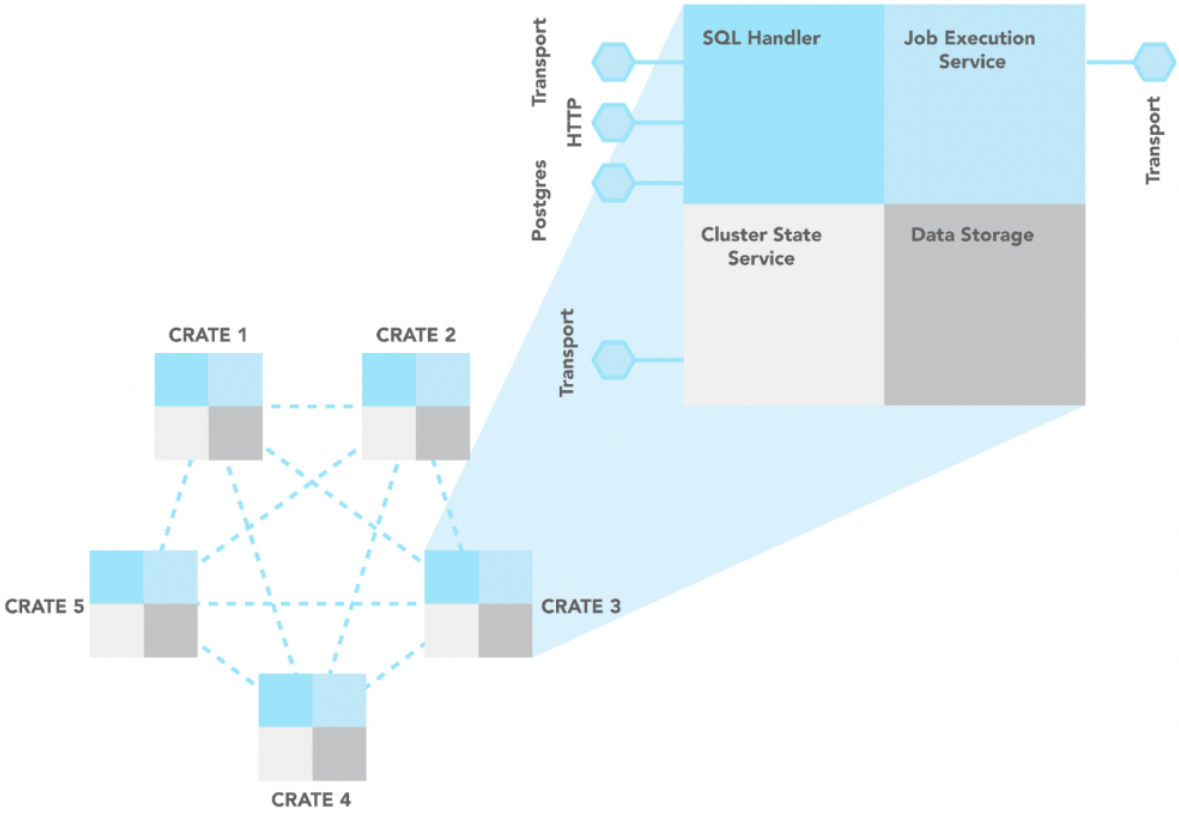
CrateDB has been designed with distribution in mind since its inception, employing a shared-nothing architecture that seamlessly scales to hundreds of nodes with minimal operational effort. This design aims to achieve:

- A cost-efficient setup on commodity hardware
- Self-healing capabilities in case of node failures
- Zero-downtime maintenance operations

Node Architecture

In contrast to a primary-secondary architecture, every node in the CrateDB cluster can perform every operation, making all nodes equal in terms of functionality and configuration. The four major components are:

- **SQL Handler:** Responsible for incoming client requests, the SQL Handler parses and analyzes SQL statement, creating an execution plan.
- **Job Execution Service:** Manages the execution of plans (“jobs”), with defined phases and resulting operations. Jobs, consisting of multiple operations, are distributed via the Transport Protocol to involved nodes, both local and remote.
- **Cluster State Service:** Manages the cluster state, including master node election and node discovery, making it a key component for cluster setup.
- **Data storage component:** Handles operations for storing and retrieving data from disk based on the execution plan. CrateDB stores data in sharded tables, dividing and storing them across multiple nodes. Each shard is a distinct Lucene index stored physically on the file system with reads and writes operating at the shard level.



Interconnected, uniform CrateDB nodes

Each node is accessible via three ports: SQL queries are accepted, and results are returned via HTTP or the PostgreSQL Wire Protocol. An additional transport port is used for inter-cluster communication.

Cluster State Management

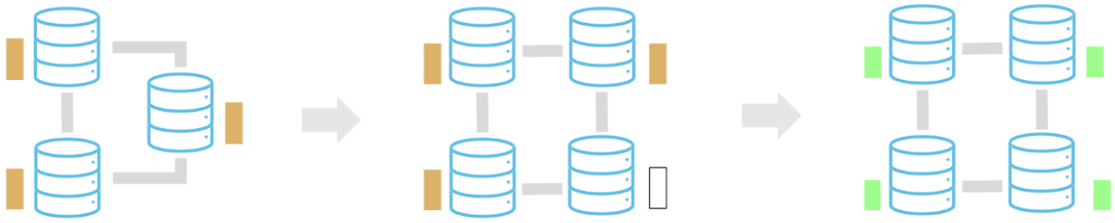
Each node in the cluster maintains a versioned copy of the latest cluster state. However, only one node in the cluster – the master node – can change the state at runtime. When the master node updates the cluster state, it publishes the new state to all nodes in the cluster and awaits responses from all nodes before processing the next update.
The cluster state encompasses all essential information for maintaining the cluster and coordinating operations, including:

- Global cluster settings
- Discovered nodes and their status
- Schemas of tables
- The status and location of primary and replica shards

At any given time, there can be only one master node. The cluster becomes available to serve requests once a master has been elected. The election, requiring a majority (also known as quorum) among master-eligible nodes, is a crucial step in this process.

Horizontal Scalability

As business demands grow, so do data volumes and hardware requirements over time. When there is an increase in requirements for CPU, RAM, and Disk Storage/IOPS, additional nodes can be seamlessly added to the CrateDB cluster without manual intervention. The cluster automatically rebalances the data to accommodate the new nodes.



Automatic Redistribution of Data when Scaling Horizontally

The diagram above illustrates the automatic redistribution process:

1. The initial three node cluster utilizes about 70% of the available storage space.
2. The addition of a new node results in an unbalanced distribution of data.
3. The automatic redistribution of data initiates, until an almost equal level of storage consumption across the four nodes is achieved again.

High Availability

One of the key benefits of a distributed database is its ability to provide high availability for always-on applications.

- CrateDB goes beyond just allowing multi-node setups; nodes can be distributed across multiple availability zones or data centers to further enhance availability.
- The system ensures uninterrupted data access during maintenance operations through the execution of rolling software updates.
- CrateDB clusters exhibit self-healing characteristics, where nodes re-joining a cluster after a failover automatically synchronize with the latest data.

Achieving high availability requires a minimum of three nodes to maintain a quorum for master node election, which holds the cluster state. The determination of the number of nodes is guided by the availability Service Level Agreement (SLA), specifying how many nodes can fail before the cluster cannot accept reads and writes. For instance:

- In a three-node cluster, one node can be offline.
- In a five-node cluster, two nodes can be offline, accommodating scenarios such as hardware failures and concurrent rolling maintenance operations.

Users have the flexibility, on a per-table level, to decide how many replicas of the data should be created. This choice dictates how many nodes each table and its shards are replicated on, providing fine-grained control over data redundancy.

It is recommended to have at least one replica; depending on the availability SLA, having two or more replicas significantly enhances the level of failure tolerance.

Failover and Recovery Process in CrateDB

The failover and recovery process typically looks like this:

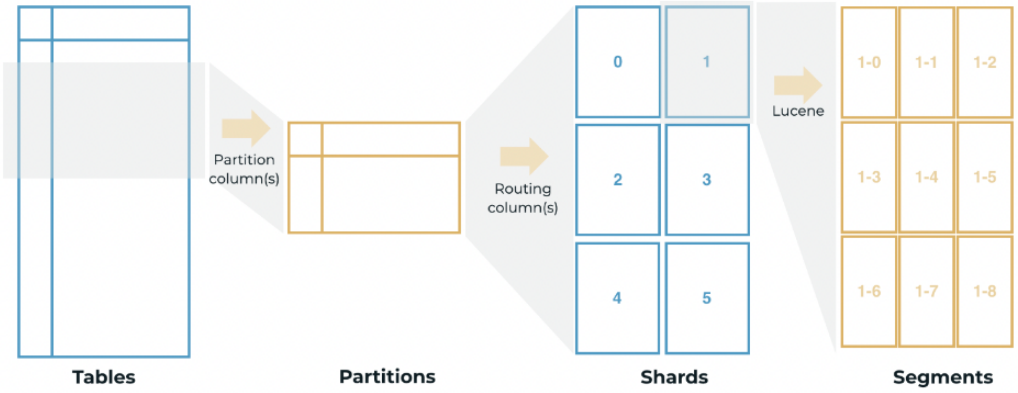
1. A node leaves the cluster due to a hardware failure, network partition, or a rolling maintenance task.
2. As data is automatically replicated in CrateDB, there is an automatic failover, ensuring data remains available despite one node leaving the cluster.
3. When a node is back up and rejoins the cluster, the data is automatically synchronized to the latest stage and rebalanced, if necessary.
4. After completing the data synchronization, the node is fully operational again, and the cluster has recovered autonomously without any manual intervention.

The following section outlines how CrateDB's sharding and replication mechanism works to ensure efficient operations.

Data Storage

In CrateDB, every table is both sharded and optionally partitioned, resulting in tables being divided and distributed across the nodes of a cluster. Each shard in CrateDB corresponds to a Lucene index made of segments stored on the file system. Physically, the files reside under one of the configured data directories of a node.

Lucene's characteristic of only appending data to segment files ensures that data written to the disk is never mutated. This characteristic simplifies replication and recovery processes, as syncing a shard is a straightforward process of fetching data from a specific marker.



Tables, Partitions, Shards, and Segments in CrateDB

CrateDB performs periodic merges of segments as they grow over time. During a merge, documents marked as deleted are discarded, and newly created segments contain only valid, non-deleted documents from the original segments. This merging process is triggered automatically in the background, and users can also execute a segment merge on demand using the OPTIMIZE TABLE command.

It's important to note that CrateDB employs the concept of soft delete to accelerate the replication and recovery process. The retention lease period specifies how long the deleted documents must be preserved to free up disk space occupied by deleted records. CrateDB discards documents only after the expiration of this period, with the default retention lease period set to 12 hours.

Partitioning and Sharding

Tables in CrateDB can be divided by defining **partition columns**.

- Each table partition is then further split into a configured number of shards, with these shards distributed across the cluster.
- As nodes are added to the cluster, CrateDB dynamically adjusts shard locations to achieve the maximum possible distribution.
- Non-partitioned tables, which function as a single partition, are still divided into the configured number of shards.
- When a record with a new distinct combination of values for the configured partition columns is inserted, CrateDB dynamically creates a new partition on the fly, inserting the document into this newly created partition.

Partitioned tables in CrateDB offer several advantages:

- **Flexible Sharding:** The number of shards can be modified on a partitioned table, influencing the number of shards used for subsequent partition creations. This flexibility allows database administrators to initially start with a few shards per partition and scale up as needed, accommodating varying traffic and ingest rates over the application's lifecycle.
- **Dynamic Scaling for Old Partitions:** For older partitions, you have the flexibility to decrease the number of shards or move them to cost-effective disks, slower machines, or other zones.
- **Granular Backup and Restoration:** Partitions can be individually backed up and restored, providing granular control over data management.
- **Improved Query Performance:** Queries with filters in the WHERE clause that identify a single partition or a subset of partitions perform better than querying all partitions. This is because shards of excluded partitions don't need to be accessed, optimizing query performance.
- **Cost-Effective Data Deletion:** Deleting data from a partitioned table is cost-effective when full partitions are dropped. Full partitions can be dropped using DELETE statements where the optimizer can infer, from the WHERE clause and partition columns, that all records of a partition match without evaluating against individual records.

Read requests in CrateDB are automatically broken down and executed in parallel across multiple shards on multiple nodes, significantly enhancing read performance. With a fixed number of primary shards, individual rows can be routed to a fixed shard number with a simple formula:

shard number = hash(routing column) mod total primary shards

When hash values are evenly distributed (which is typically the case), rows will be evenly distributed amongst the fixed number of available shards.

The routing column, specified during table creation, determines the shard allocation. All rows with the same value in the routing column are stored in the same shard. If a primary key is defined, it serves as the default routing column; otherwise, the internal document ID is used. If the routing column is explicitly defined, it must match a primary key column.

Replication

Replication in CrateDB operates at the table level, ensuring that each table shard has the configured number of copies available. In this setup, there is always one primary shard, and the remaining shards are designated as replica shards.

- Write operations are directed exclusively to the primary shard, while read operations can be distributed across any shard.
- CrateDB efficiently synchronizes data from the primary shard to all replica shards, benefiting from the append-only characteristic of Lucene segments.
- In the event of a primary shard loss due to node failure, CrateDB automatically promotes a replica shard to primary status.

Replication not only enhances fault tolerance but also improves read performance by increasing the number of shards distributed across a cluster. This increase provides more opportunities for CrateDB to parallelize query execution across multiple nodes.

The figures below illustrate the concept, with a table split into multiple shards and distributed evenly across a cluster. The first example shows table t1 with shards s1 to s3:

```
CREATE TABLE t1 (
  name STRING
) CLUSTERED INTO 3 SHARDS;
```

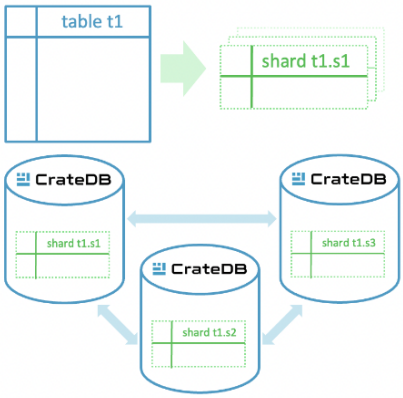


Table with Three Shards Distributed Across Nodes

Data is replicated at a shard level, as depicted in the second figure showing table t1 with three shards (s1 to s3) and one replica (r1 to r3) each:

```
CREATE TABLE t1 (
  name STRING
) CLUSTERED INTO 3 SHARDS
  WITH ("number_of_replicas" = '1');
```

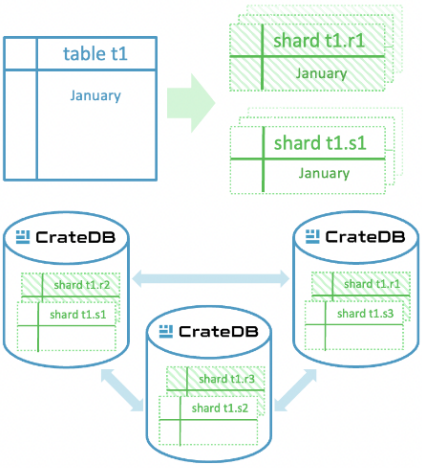
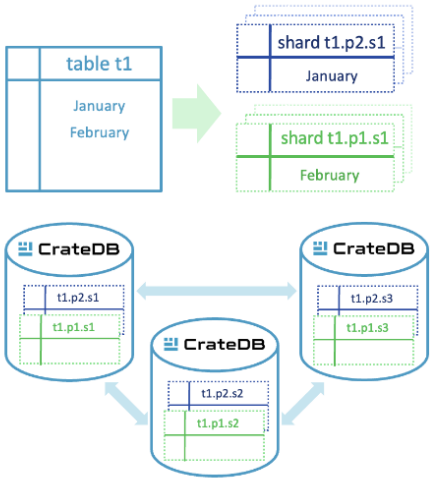


Table with Three Shards and One Replica Each Distributed Across Nodes

The third illustration demonstrates a table split into two partitions, using the month as a routing table. Adding data for more months automatically creates a new partition:

```
CREATE TABLE t1 (
  name STRING,
  month TIMESTAMP
) CLUSTERED INTO 3 SHARDS
  PARTITIONED BY (month);

INSERT INTO t1 (name, month) VALUES (
 ('foo', '2023-01-01'),
 ('bar', '2023-02-01')
);
```



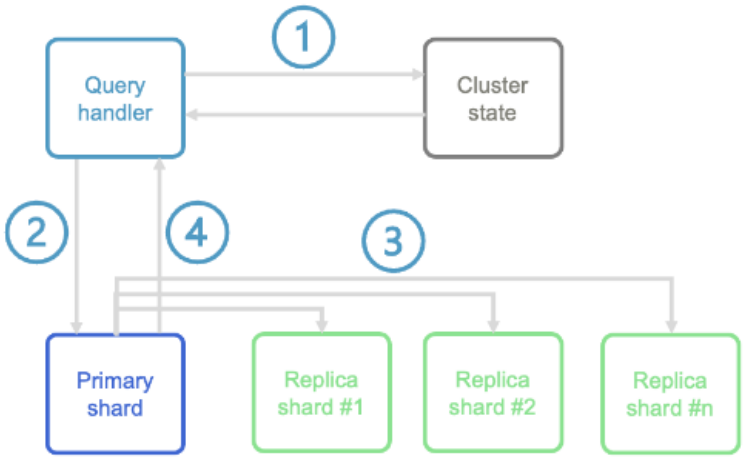
Partitioned Table with Three Shards per Partition

Read operations are executed on primary shards and their replicas to increase query distribution and enhance performance. CrateDB randomly assigns a shard when routing an operation, with the option to configure this behavior.

Write operations follow a synchronous process across all active replicas:

1. Look up the primary shard and active replicas in the cluster state. For this step to succeed, both the primary shard and a quorum of the configured replicas must be available.
2. Route the operation to the primary shard for execution.
3. Execute the operation on the primary shard.
4. If the operation succeeds on the primary shard, execute the operation on all replicas in parallel.
5. Return the operation result to the caller after all replica operations are finished.

Any replica shard failing to write data or timing out in step 5 is immediately considered as unavailable.



Writing and Replication in CrateDB

Atomicity, Durability, and Consistency

While CrateDB defaults to an eventually consistent database, it enforces the principles of atomicity, durability, and consistency:

Atomicity at Row/Document Level:

- Each table row in CrateDB is a semi-structured document with arbitrary nesting through the use of object and array types.
- Operations on documents are atomic, ensuring that a write operation on a document either succeeds or has no effect. This holds true irrespective of the nesting depth or size of the document.
- CrateDB does not provide transactions, but every document has an assigned version number that increases with each change. Patterns like Optimistic Concurrency Control offer a solution to check if changes on multiple rows have been applied.

Durability:

- Every shard maintains a Write Ahead Log (WAL) also known as translog, ensuring that document operations are persisted to disk without requiring a Lucene commit for every write.
- When the translog is flushed, all data is written to Lucene's persistent index storage, and the translog is cleared.
- In case of an unclean shard shutdown, transactions in the translog are replayed upon startup to ensure all executed operations are permanent.
- The translog is directly transferred during new replica initialization, eliminating the need to flush segments to disk solely for replica recovery purposes.

Consistency Levels:

- CrateDB accesses documents internally through 'get' queries by primary key and 'search' queries by any secondary attribute, transparent to the user.
- **For search operations, CrateDB adopts an Eventual Consistency model** rooted in the concept of shared IndexReaders, which provide caching and reverse lookup capabilities for shards. An IndexReader is bound to the Lucene segment it originated from and requires periodic refreshes to reflect new changes. This refresh can occur either in a time-based manner, configurable by the user, or manually triggered using the REFRESH TABLE command. Therefore, a search operation only recognizes a change if the corresponding IndexReader was refreshed after the change occurred.
- In contrast, **Strong Consistency is ensured for operations by the primary key** when the shard routing key and row identifier can be computed from query specification, such as when the full primary key is defined in the WHERE clause. This approach, accessing a document through a single shard and a straightforward index lookup on the '_id' field, proves to be the most efficient method.

If a query specification results in such a 'get' operation, changes are immediately visible. This is achieved by first looking up the document in the translog, which always holds the most recent version of the document. This mechanism enables the common 'fetch' and 'update' use case. For instance, if a client updates a row and subsequently looks up that row by its primary key, the changes will be immediately visible, retrieved directly from the translog without requiring a refresh of the IndexReader.

Advanced Indexing

When ingesting data, the queries you will make over time may not be clear initially, and use cases tend to evolve, leading to new requirements and query patterns.

To ensure fast query responses for any query type, CrateDB automatically indexes every attribute by default. Depending on the data type, the following strategies are applied:

- **Inverted Index for text values**: Facilitates efficient search for precise text matches, including support for wildcards and regular expressions.
- **Block k-d trees (BKD) for numeric, date, and geospatial values:** Highly efficient indexes designed for optimal IO. Most data structure resides in on-disk blocks, with a small in-heap binary tree index structure for locating blocks at search time. This design ensures excellent query and update performance regardless of the number of updates performed.
- **HNSW (Hierarchical Navigable Small World) graphs for high dimensional vectors:** Enables efficient approximate nearest neighbor search, commonly known as similarity search.

Additionally, **full-text indexes** can be added on-demand to unlock features like fuzzy search, phrase search, and attribute boosting. CrateDB offers over 30 languages, 11 analyzers, 15 tokenizers, more than 35 token filters, and the flexibility for custom analyzers and tokenizers.

Columnar Storage

In conjunction with advanced indexing strategies, CrateDB adopts a columnar storage approach that facilitates fast queries and complex aggregations across large data sets:

- Storing data for the same field together **optimizes file-system cache utilization**. This design eliminates the need to load unnecessary data for fields not needed by the query.
- By segmenting data into blocks and incorporating metadata about the range or set of unique values in the block header, certain queries may entirely **skip unnecessary blocks during execution**.
- Implementation of specific techniques allows **querying data without decompressing it first**.

Data Tiering: Hot, Warm, and Cold Data

Data tiering is crucial for cost-effective management of large datasets, offering the following benefits:

- **Cost-Effective Data Management:** In CrateDB, data tiering efficiently manages data across various storage tiers based on access frequency and performance needs. 'Hot' data, frequently accessed, can reside on high-performance storage, while 'cold' data, accessed less frequently, can be stored on more cost-effective solutions.
- **Automatic and Manual Tiering:** CrateDB provides support for both automatic and manual data tiering, allowing administrators to define policies for data movement between tiers or to manually relocate data based on specific requirements.
- **Optimized Resource Utilization:** Leveraging data tiering enables organizations to significantly reduce costs and optimize resource utilization. This approach ensures that high-cost resources are reserved for critical, high-performance workloads, while lower-cost storage is employed for less demanding tasks.

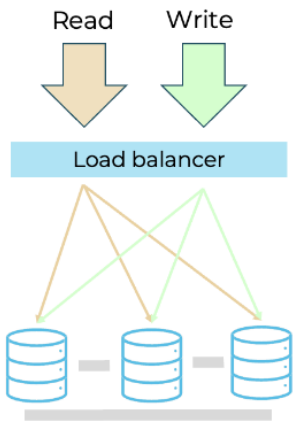
Data Tiering in CrateDB

Performance: High-Volume Concurrent Reads and Writes

Distributed Query Engine

The capability to efficiently manage extensive concurrent reads and writes is crucial. CrateDB's query engine has been engineered from the outset to meet this requirement, ensuring that users can extract insights and perform actions on vast datasets with unparalleled speed and efficiency.

We will explore how CrateDB's query engine is architected to optimize data throughput and query performance, particularly as the number of concurrent operations grows. Key features, including **distributed query processing**, **advanced indexing techniques**, **real-time data ingestion**, **and real-time querying**, synergize to deliver a seamless and high-performance user experience.



High-Volume Concurrent Reads and Writes in CrateDB

Distributed Writes

At its core, CrateDB employs a **sharding mechanism** to distribute data across multiple nodes in a cluster. Upon data ingestion, it undergoes sharding, breaking it into self-contained subsets known as shards. This approach optimally distributes the workload across available CPUs, even on single node.

The sharding strategy enables parallel write operations, allowing independent and concurrent writing to each shard on different nodes. This distribution prevents any single node from becoming a bottleneck, improving **write throughput and scalability.**

Each shard in CrateDB is associated with a **Lucene index**, a structure storing data in immutable segments. When new data is written, it first goes to an in-memory buffer. Upon reaching a specific size, this data is flushed and appended to a new Lucene segment on disk. This append-only approach ensures efficient write operations without the need to rewrite or rearrange existing data or table-level locks.

As more data is written, segments accumulate in a shard. CrateDB employs **periodic segment merging** as a background operation to optimize storage and performance. Merging smaller segments into larger ones reduces the overhead of managing multiple segments and enhances read efficiency, as fewer segments need to be scanned during read operations. The process is carefully tuned to balance write load, storage efficiency, and read performance, leveraging Lucene's advanced algorithms to avoid bottlenecks.

In addition to sharding, CrateDB **replicates** data at the shard level across various nodes, ensuring data availability and fault tolerance. Write operations are deemed successful only when the data is written to the primary shard and replicated to the required number of replica shards, as per the defined replication factor. This replication, coupled with the distributed nature of sharding, contributes to the robustness and reliability of write operations in CrateDB.

Distributed Reads

In addition to the advanced mechanisms mentioned earlier, such as columnar storage, efficient indexing per datatype, full-text indexes, and similarity search, achieving outstanding performance in CrateDB involves the distribution of reads and the lock-free execution of writes and reads.

CrateDB’s design, focused on distribution, leads to operations being split across shards and their replicas by the query planner. This strategy accelerates aggregations by selecting only the necessary data partitions, utilizing available hardware on individual nodes, distributing queries across all nodes, and pushing down aggregations to multiple nodes to reduce pressure during the merge step of query execution.

Executing exact aggregations in distributed databases presents challenges but holds significant potential to enhance query speed. CrateDB's approach to aggregation involves four steps: **collect**, **reshuffle**, **aggregate**, and **merge**. The reshuffling step occurs in CrateDB's unique distribution layer, redistributing intermediate responses between nodes to expedite processing. While each request is sent to all nodes, the responses pass through the distribution layer. Each node hashes the returned values and distributes each row, along with the values, to other nodes based on the hash value. Each node can then reduce its distinct part of the data to the aggregated values, sometimes even to a single value, as it knows that no other node has data with the same values. This innovative approach helps optimize the execution of aggregations in a distributed environment.

Security

Data encryption

CrateDB provides the capability to encrypt internal communication between CrateDB nodes and external communication with HTTP and PostgreSQL clients using Transport Layer Security (TLS).

SSL can be configured on a per-protocol basis:

- If SSL is enabled for HTTP, all connections will require HTTPS.
- For the PostgreSQL wire protocol, by default, SSL negotiation is possible on a per-connection basis. However, SSL enforcement can be configured via Host-Based Authentication.
- If SSL is enabled for the CrateDB transport protocol (used for intra-node communication), nodes will only accept SSL connections (ssl.transport.mode set to on). This ensures encrypted communication between nodes, particularly important when installed across zones or data centers.

CrateDB strongly recommends the use of encrypted disks for storing data, and this is the default setting in CrateDB's managed cloud offering. This emphasis on encryption helps enhance the overall security and confidentiality of data in transit and at rest within the CrateDB environment.

Authentication

Comprehensive authentication mechanisms are in place to ensure secure access:

- **User Authentication:** Authentication is required to access the database. It can be done by submitting a username/password or using a client certificate.
- **TLS for Secure Connections:** Connections can be secured through Transport Layer Security (TLS). This is crucial for ensuring encrypted communication between clients and the CrateDB server.
- **Password Security:** Passwords are securely stored with a per-user salt and hashed using the PBKDF2 key derivation function and the SHA-512 hash algorithm.
- **Host-Based Authentication:** Host-based authentication in CrateDB adds an extra layer of security by verifying the username, IP address, protocol, and connection scheme. Trust-based authentication is recommended for secure network environments and during development.
- **Node-to-Node Authentication:** Internal node-to-node communication is secured either through trust-based authentication or by using certificates.

Authorization

CrateDB employs Role-Based Access Control (RBAC) to manage user access and privileges.

- RBAC is used to associate users with specific roles.
- Each role is assigned specific privileges, defining the degree of access to various resources within CrateDB.
- Privileges can be granted at different levels, including the cluster, schema, and table/view levels.

Different types of privileges can be assigned to users and roles:

- **Data Query Language (DQL) Privilege:** Allows users/roles to execute SELECT, SHOW, REFRESH, and COPY TO statements. Enables the use of available user-defined functions on the specified objects.
- **Data Manipulation Language (DML) Privilege:** Allows users/roles to execute INSERT, COPY FROM, UPDATE, and DELETE statements on the specified objects.
- **Data Definition Language (DDL) Privilege:** Allows users/roles to execute statements for creating, altering, and deleting tables, views, functions, repositories, and snapshots.
- **Administration Language (AL) Privilege:** Grants users/roles the ability to create and drop users and roles. Enables the assignment of privileges and modification of global settings.

Auditing

Queries performed on CrateDB are logged in internal tables and can be exported to log files on disk for further analysis or integration into SIEM systems. This allows administrators to ensure that system management tasks are being appropriately performed. Monitoring information can be collected and reported using JMX and forwarded, for example, to Prometheus, a popular cloud-native monitoring tool.

ISO 27001 Certification

CrateDB has achieved the ISO/IEC 27001:2013 certification for its CrateDB Cloud SaaS offering. This certification signifies the attainment of the highest standards in operational and information security, emphasizing a commitment to safeguarding the valuable information of users. By aligning the information security management system (ISMS) with the ISO/IEC 27001 standard, CrateDB ensures the highest assurance of confidentiality, integrity, and availability for sensitive information.

Flexible Deployment Models

In today's diverse technological landscape, the one-size-fits-all approach to database deployment is no longer sufficient. Enterprises have unique requirements based on their operational, security, and regulatory needs. Recognizing this diversity, CrateDB offers a **range of flexible deployment options**, ensuring that regardless of your environment – whether it's at the Edge, on-premises, hybrid, or in the cloud – there is a solution that aligns seamlessly with your infrastructure and business goals.

By incorporating **zone-aware deployments** and **data tiering**, CrateDB delivers a highly adaptable and efficient database solution capable of meeting the nuanced demands of modern, data-driven enterprises. These features underscore CrateDB's commitment to providing versatile, performance-oriented, and cost-effective data management solutions in various deployment scenarios.

Containerized

CrateDB facilitates seamless deployment and scaling in diverse environments by offering support for containerization and orchestration tools such as Docker and Kubernetes. This enhances operational flexibility and ensures adaptability to varying infrastructure requirements.

Private & Public Cloud

- **Private Cloud:** For those seeking cloud-like flexibility while maintaining data control within their own environment, CrateDB can be deployed in private clouds. This approach combines the advantages of cloud computing with the privacy and security controls of on-premises deployment.
- **Public Cloud:** CrateDB excels in public cloud environments, including AWS, Azure, and GCP. This deployment option is ideal for businesses in pursuit of scalability and flexibility. Leveraging the public cloud model enables CrateDB to dynamically scale resources based on workload demands, providing cost-effective solutions without requiring upfront hardware investments.

On-Prem & Hybrid

- **Full Control and Security:** Organizations prioritizing security and control can deploy CrateDB on-premises. This configuration enables businesses to utilize their existing infrastructure, maintaining comprehensive control over data and database management in adherence to stringent compliance and security protocols.

- **Best of Both Worlds:** Hybrid deployments offer a balanced solution, combining the control and security of on-premises systems with the scalability and flexibility of cloud environments. CrateDB seamlessly integrates with this model, allowing sensitive data to be stored on-premises while leveraging cloud resources for scalable computing and storage.

Edge

- **Optimized for real-time analytics and Edge AI:** CrateDB is uniquely positioned for edge computing environments, especially in IoT and real-time analytics scenarios. Deploying directly at the Edge, CrateDB facilitates immediate data processing, reducing latency and bandwidth constraints associated with data transfer to centralized systems. This ensures faster insights and decision-making at the point of data generation.

Logical Replication Between Clusters

Logical replication is a method of replicating data across multiple clusters. In CrateDB, this process operates on a publish and subscribe model, where subscribers pull data from the publications of the publisher they subscribed to. Replicated tables on a subscriber can further be published to other clusters, enabling the chaining of subscriptions.

Logical replication proves beneficial in various use cases:

- **Consolidating data for Aggregated Reports:** Ideal for aggregating data from multiple clusters to generate comprehensive reports.
- **Ensuring High Availability:** Logical replication facilitates high availability by redistributing data if one cluster becomes unavailable.
- **Replicating Across Different CrateDB Versions:** Logical replication supports replication between different compatible versions of CrateDB. However, replicating tables created on a cluster with a higher major/minor version to a cluster with a lower major/minor version is not supported.

Zone Aware Deployments

Zone aware deployments offer numerous advantages through the integrated capabilities of CrateDB:

- **Geo-Distributed Architecture:** In geo-distributed environments, zone-aware deployments in CrateDB are pivotal. This feature empowers the database to be cognizant of the physical location of its nodes across various regions or data centers.

- **Improved Performance and Reduced Latency:** CrateDB leverages its understanding of the geographical distribution of data to route queries and manage data replication, minimizing latency intelligently. This is particularly advantageous for global applications that necessitate fast data access from specific geographic locations.
- **Enhanced Resilience:** Zone-aware deployments significantly contribute to the resilience of the database. By distributing data across multiple zones, CrateDB ensures high availability and durability, safeguarding against data center outages or regional disruptions.

Zone-Aware Deployments of CrateDB

Ecosystem Integrations

In the dynamic landscape of data management and analytics, a database's capability to seamlessly integrate with various components of the technology ecosystem is crucial. This section outlines how CrateDB fits into a broader technology ecosystem, highlighting its strengths in both ingesting and feeding data. It emphasizes the adaptability and openness of CrateDB in integrating with various tools and platforms.

Programming Languages

CrateDB extends support to a range of programming languages through dedicated drivers and integrations.

- **Java Integration:** For Java, CrateDB provides a JDBC driver, ensuring seamless integration with numerous Java applications and frameworks. This makes it the optimal choice for Java developers seeking to harness CrateDB's scalability and performance.
- **Python Integration:** In the realm of Python, a language prominent in data analytics and machine learning, CrateDB offers a specialized Python driver and an SQLAlchemy dialect. This driver facilitates smooth integration with Python's extensive ecosystem of data processing and analytics libraries, positioning CrateDB as an appealing option for data scientists and Python developers.
- **.NET Compatibility:** .NET can utilize the Npgsql driver to connect and interact with CrateDB, leveraging familiar interfaces within their applications.
- **Web Development and Scripting Languages:** For web development and scripting languages such as Node.js, PHP, Ruby, CrateDB is accessible through PostgreSQL-compatible drivers. Utilizing PostgreSQL Wire Protocol enables developers in these languages to interact with CrateDB, capitalizing on the familiarity and stability of established drivers.
- **Go Programming Language:** CrateDB supports the Go programming language through the PostgreSQL Wire Protocol, ensuring efficiency in building high-performance applications. This compatibility enables Go developers to seamlessly incorporate CrateDB into their applications, benefiting from its distributed architecture and real-time data processing capabilities.
- **RESTful HTTP Interface:** CrateDB offers a RESTful HTTP interface, providing a versatile and straightforward option for virtually any programming language capable of making HTTP requests. This ensures accessibility even in languages where specific CrateDB drivers are not available.

SQL- and API-based Integrations

CrateDB provides SQL-based access through the PostgreSQL Wire Protocol, rendering it compatible with any tool or application supporting SQL queries. Additionally, its **HTTP interface** facilitates API integrations by allowing the execution of SQL statements over HTTP, ensuring versatile and seamless interaction with various applications and services.

Data Ingestion & Streaming

CrateDB excels in real-time data ingestion, integrating with streaming platforms such as Apache Kafka. This capability enables efficient handling of data streams originating from IoT devices and other sources, proving essential for real-time analytics.

Data Processing, ETL and Orchestration

CrateDB seamlessly integrates with a variety of data processing tools, including Apache NiFi, Apache Flink, dbt, and ETL tools like Talend and StreamSets. This integration empowers efficient data transformation and movement, which is crucial for managing large datasets and preparing data for analysis.
When orchestrating data integration and processing workflows, CrateDB performs optimally when paired with platforms such as Prefect, Airflow, and others like NodeRED or n8n.

Data Visualization & Analytics

CrateDB integrates with popular BI and visualization tools like Grafana, Apache Superset, Tableau, and Power BI. This integration empowers users to craft insightful visual representations of data, facilitating thorough analysis and informed decision-making.

Monitoring & Logging Tools

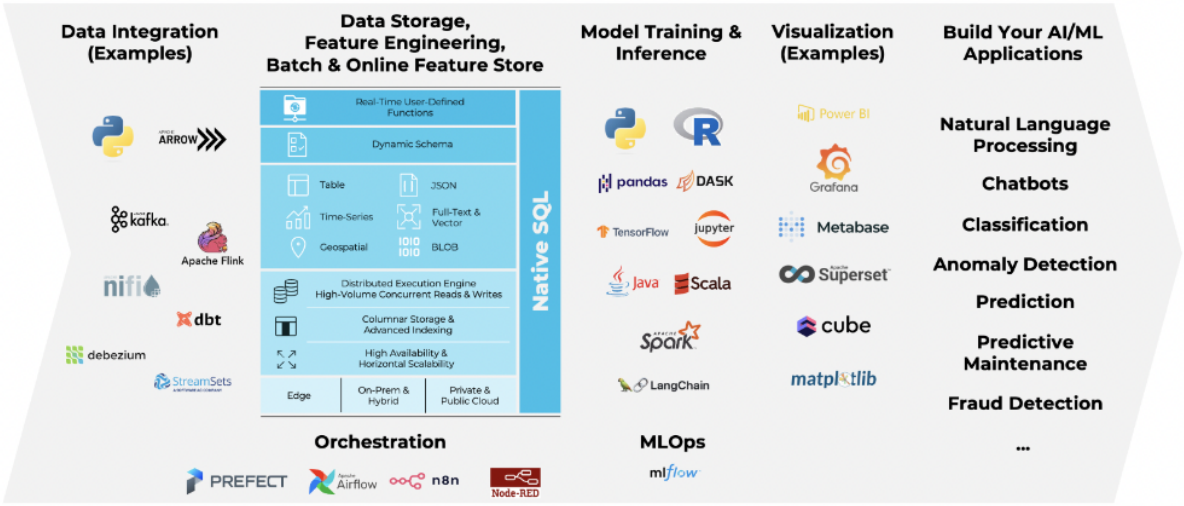
CrateDB can integrate with monitoring and logging tools, enabling efficient tracking of database performance and operational metrics. This capability is crucial for maintaining system health and performance, ensuring proactive management and optimization.

AI and Machine Learning Applications

CrateDB is also highly beneficial for AI and Machine Learning applications. Its real-time data processing capability makes it an excellent choice for storing and analyzing the vast datasets commonly used in machine learning. The distributed SQL query engine allows for fast data exploration, and the full-text search capabilities are useful for natural language processing tasks. Additionally, the support for geospatial queries makes it versatile for AI applications requiring location data.

With its Python client library, CrateDB offers an intuitive interface for direct interaction with databases from Python scripts, facilitating the training, evaluation and deployment of machine learning models. This includes reading data into data frames for **preprocessing**, executing SQL queries for **data manipulation**, and writing **predictions** back to the database. The combination of CrateDB and Python streamlines the workflow for data scientists, allowing them to focus more on model development and less on data management.

Moreover, CrateDB is an ideal backend for **model tracking with MLflow**, providing a robust solution for managing the entire machine learning lifecycle. Its seamless integration with orchestration tools further enables the creation of **end-to-end workflows**.



Getting started with CrateDB

CrateDB Cloud & Free Tier

Begin your journey with CrateDB by exploring CrateDB Cloud. This offers a convenient way to experience CrateDB's capabilities right away with nothing to install. Access tutorials and guides on http://cratedb.com to get hands-on experience with the cloud platform.

To start your free tier cluster: https://console.cratedb.cloud/

Community & Learning Resources

For in-depth understanding, utilize the extensive range of documentation, how-to guides, and community resources available online. These resources cover various aspects of working with CrateDB, from basic setup to advanced usage: https://docs.cratedb.com/

In addition to the official resources, CrateDB has a vibrant and supportive community. Engaging with the community is a great way to get insights, tips, and real-world examples of CrateDB implementations. Whether you're encountering a technical challenge or seeking innovative ways to use CrateDB, the community is a valuable resource for shared knowledge and collaborative problem-solving: https://community.cratedb.com/

Speak to a Technical Expert

If you need more tailored advice or have specific questions, don't hesitate to contact the sales team or speak to a technical expert. They can provide personalized guidance and answers to your queries. Book a session directly on https://cratedb.com/

CrateDB is an open source, multi-model, and distributed database that offers high performance, scalability, and flexibility. Our team is on a mission to develop reliable and scalable database technology where response time is never an issue, regardless of the complexity and volume of data.

Contact us to learn more, or visit cratedb.com to understand how we can help with your data needs.

CrateDB