# MongoDB.

# FACT SHEET

## DATA MODEL AND SCHEMA

- MongoDB is a **document-based data model**
- Data is stored in BSON (Binary JSON), which allows for a *schema-less design*
- Documents consist of key-value pairs
- Supports complex data structures like arrays and nested documents
- Unlike relational databases, collections do not enforce a fixed schema

## CONSISTENCY AND REPLICATION

**Theorem & Consistency**
MongoDB is a NoSQL database and follows CAP theorem principles:

*Partition Tolerance (P):*
- Built to work in distributed environments

*Consistency vs. Availability Trade-off:*
- Strong consistency when reading from the primary node
- Eventual consistency when reading from secondary nodes

*Tunable Consistency:*
- *D*evelopers can choose read and write concerns to balance consistency and availability
- "Majority" read concern ensures data consistency across multiple nodes

## SECURITY

**- Authentication & Authorization:**
- **S**upports username/password authentication
- Integrates with LDAP and x.509 certificate-based authentication
- Role-Based Access Control (RBAC) for fine-grained permissions

**- Encryption:**
- **E**ncryption at Rest: Protects stored data
- Encryption in Transit: Uses TLS/SSL to secure client-server communication

**- Auditing & Compliance:** Provides detailed logging for security monitoring

## SPECIFIC USE CASES

- ***Rapid Development & Schema Flexibility:***

Ideal for startups and evolving applications

- **B*ig Data & Real-Time Analytics:***

Supports high-velocity data processing

- ***Content Management Systems*:**

Stores complex and unstructured data

- **I*oT & Mobile Applications:***

Handles large-scale, high-speed data ingestion

- ***Geospatial & Search Applications:***

Provides built-in geospatial indexing and full-text search

## EXTRA FEATURES

- **Indexing**: Supports various types of indexes (compound, geospatial, text, etc.) to optimize queries
- **Aggregation Framework**: Provides a pipeline-based system for data transformation and analysis
- **Flexible Query Language**: Supports rich queries, including filtering, sorting, and regex searches
- **Change Streams**: Enables real-time data updates for applications
- ***Replication and High Availability***: Uses replica sets for failover and data redundancy

# CLUSTERS

**Replica Set**

- A replica set is a group of MongoDB servers that store identical copies of data.
- Purpose:
  - *High availability* (data is always accessible).
  - *Redundancy* (backup in case a server fails).
  - Handles failovers and maintenance with minimal downtime.
- Can handle read operations.

**Sharded Cluster**

- Also known as horizontal scaling.
- Data is split and distributed across multiple servers (shards).
- Purpose:
  - Scales read and write operations.
  - Useful when dealing with large datasets or high traffic.

# HISTORY

In **2009**, MongoDB was officially released as an open-source project, allowing developers worldwide to leverage its features for free. The database quickly gained popularity due to its document-oriented design, scalability, and flexibility. Unlike traditional SQL databases, MongoDB provided a schema-less structure, making it particularly well-suited for rapidly evolving applications and big data workloads.

**Economic information**

MongoDB reported $2.01 billion in revenue for fiscal year 2025, with its cloud service Atlas making up 68% of that, showing strong growth in the cloud database market. It holds a leading position among NoSQL databases and is used by major companies like Adobe, eBay, and Coinbase.

# OPLOG AND ELECTION PROCESS

**Oplog:**

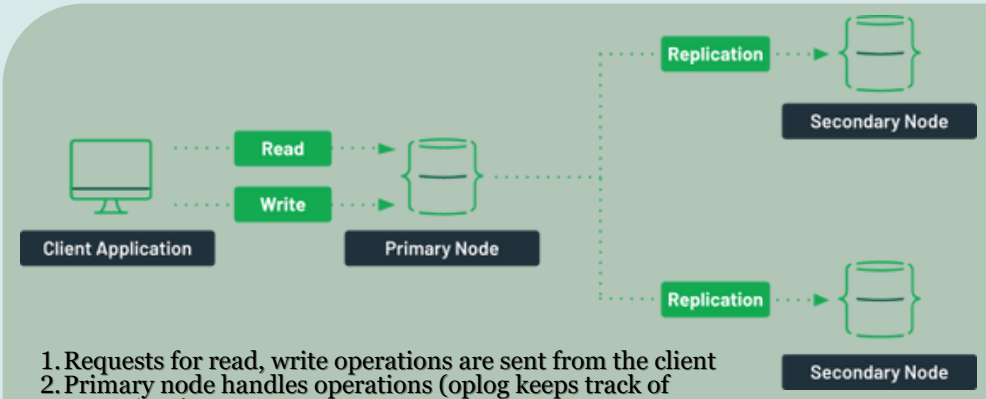
- special capped collection that stores a rolling log of operations.
- Used to sync secondary nodes with the primary.
- Dynamically resizes to avoid deleting important commits prematurely.

**Election Process:**

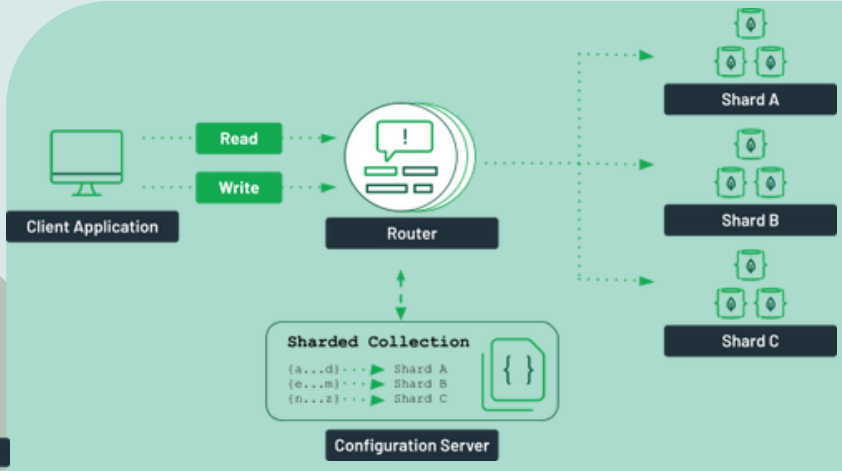
- Triggered when the primary becomes unavailable.
- A new primary is elected from secondary nodes.

*Triggers include:*

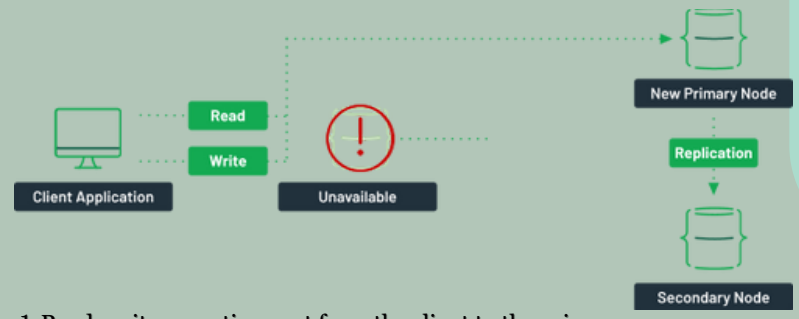
- Node failure
- Timeout >10 seconds (default)
- Adding new nodes
- Initial setup or planned maintenance




1. Requests for read, write operations are sent from the client
2. Primary node handles operations (oplog keeps track of operations)
3. Operations replicated in secondary nodes using oplog




1. Read, write operations sent to the router
2. Router decides which shard contains the relevant information
3. Requests forwarded to the correct shard




1. Read, write operation sent from the client to the primary node
2. Primary node is detected as not working
3. Secondary node is elected as the new primary node (election process)
4. Operation is executed on the new primary node

mongo DB