UNIT III NOSQL DATABASES 9

NoSQL – CAP Theorem – Sharding - Document based – MongoDB Operation: Insert, Update, Delete, Query, Indexing, Application, Replication, Sharding–Cassandra: Data Model, Key Space, Table Operations, CRUD Operations, CQL Types – HIVE: Data types, Database Operations, Partitioning – HiveQL – OrientDB Graph database – OrientDB Features

---

MongoDB Operation:

MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling. MongoDB is a document-oriented database. It is an open source product, developed and supported by a company named 10gen. MongoDB is a scalable, open source, high performance, document-oriented database." MongoDB was designed to work with commodity servers. Now it is used by companies of all sizes, across all industries.

MongoDB Advantages

- MongoDB is schema less. It is a document database in which one collection holds different documents.
- There may be differences between the number of fields, content and size of the document from one to another.
- Structure of a single object is clear in MongoDB.
- There are no complex joins in MongoDB.
- MongoDB provides the facility of deep query because it supports a powerful dynamic query on documents.
- It is very easy to scale.
- It uses internal memory for storing working sets and this is the reason of its fast access.

Distinctive features of MongoDB

- Easy to use
- Light Weight
- Extremely faster than RDBMS

Where MongoDB should be used

- Big and complex data
- Mobile and social infrastructure
- Content management and delivery
- User data management
- Data hub

MongoDB Data Types

MongoDB supports many data types. Some of them are:

1. **String**: String is the most commonly used datatype to store the data. It is used to store words or text. String in MongoDB must be UTF-8 valid.
2. **Integer**: This data type is used to store a numerical value. Integer can be 32-bit or 64-bit depending upon the server.
3. **Boolean**: This data type is used to store a Boolean (true/ false) value.
4. **Float**: This data type is used to store floating point values.
5. **Min/Max keys**: This data type is used to compare a value against the lowest and highest BSON elements.
6. **Arrays**: This data type is used to store arrays or list or multiple values into one key
7. **Timestamp**: This data type is used to store the data and time at which a particular event occurred. For example, recording when a document has been modified or added
8. **Object**: This datatype is used for embedded documents
9. **Null**: This data type is used to store a Null value.
10. **Symbol**: This datatype is used identically to a string; however, it's generally reserved for languages that use a specific symbol type
11. **Date**: This datatype is used to store the current date or time in UNIX time format. We can specify our own date time by creating an object of Date and passing day, month, year into it.
12. **Object ID**: This datatype is used to store the document's ID
13. **Binary data**: This datatype is used to store binary data.
14. **Code**: This datatype is used to store JavaScript code into the document.
15. **Regular expression**: This datatype is used to store regular expressions.

MongoDB Create Database

There is no create database command in MongoDB. Actually, MongoDB does not provide any command to create a database.

How and when to create database

If there is no existing database, the following command is used to create a new database.

Syntax:

use DATABASE_NAME

INPUT:-

>>>use inventory

OUTPUT:-

switched to db inventory

MongoDB Create Collection

In MongoDB, db.createCollection(name, options) is used to create collections. But usually it doesn't need to create a collection. MongoDB create collection automatically when you insert some documents.

Syntax:

db.createCollection(name, options)

Name: is a string type, specifies the name of the collection to be created.

Options: is a document type, specifies the memory size and indexing of the collection. It is an optional parameter.

Insert Operation:-

It is used to add one or more documents to the collection. It has two types,

insertOne-used to add only one document to the collection.

insertMany-used to add more than one document to the collection.

Syntax:-insertOne():-

db.collection.insertOne(<document>,{ writeConcern: <document> })

SAMPLE QUERY:-

INPUT:-

>>>db.inventory.insertOne({ item: "canvas", qty: 100, tags: ["cotton"], size: { h: 28, w: 35.5, uom: "cm" } })

OUTPUT:-

{
“acknowledge”:true,
“insertedId”:ObjectId(“603e3d2f6b88c382606523ad”)
}

Syntax:-

insertMany():-

db.collection.insertMany(
[<document 1> , <document 2>, ...],
{
writeConcern: <document>,
ordered: <boolean>
}
)

SAMPLE QUERY:-

INPUT:-

>>>db.inventory.insertMany([
{ item: "journal", qty: 25, tags: ["blank", "red"], size: { h: 14, w: 21, uom: "cm" } },
{ item: "mat", qty: 85, tags: ["gray"], size: { h: 27.9, w: 35.5, uom: "cm" } },
{ item: "mousepad", qty: 25, tags: ["gel", "blue"], size: { h: 19, w: 22.85, uom: "cm" } }
])

OUTPUT:-

{
“acknowledge”:true,
“insertedIds”:[ObjectId(“603ee5973b41040c0b3227107”),
ObjectId(“603ee5973b41040c0b3227108”)
ObjectId(“603ee5973b41040c0b32271079”)
}

READ OPERATION:-

It is used to retrieve documents from the collection based on some constraints.

Syntax:-

db.collection.find(query, { <field1>: <value>, <field2>: <value> ... })

SAMPLE QUERY:-

INPUT:-

>>>db.inventory.find({})

OUTPUT:

{“_id”: ObjectId(“603e3d2f6b88c382606523ad”),”item”: "canvas", “qty”: 100, “tags”: ["cotton"], “size”: { “h”: 28, “w”: 35.5, “uom”: "cm" } }
{ “_id”:ObjectId(“603ee5973b41040c0b327107”),”item”: "journal",” qty”: 25,” tags”: ["blank", "red"], “size”: { h: 14, w: 21, uom: "cm" } }
{“_id”: ObjectId(“603ee5973b41040c0b327108”),”item”: "mat", “qty”: 85,”tags”: ["gray"], “size”: { h: 27.9, w: 35.5, uom: "cm" } }
{ “_id”:ObjectId(“603ee5973b41040c0b3271079”),”item”: "mousepad", “qty”: 25,” tags”: ["gel", "blue"],” size”: { h: 19, w: 22.85, uom: "cm" } }

SAMPLE QUERY:-

INPUT:-

>>>db.inventory.find({qty:85})

OUTPUT:-

{“_id”: ObjectId(“603ee5973b41040c0b327108”),”item”: "mat", “qty”: 85,”tags”: ["gray"], “size”: { h: 27.9, w: 35.5, uom: "cm" } }

UPDATE OPERATION:-

It is used to modify (add/replace)one or more documents in the collection. It consists of 3 types:

updateOne

update many

Syntax:-

db.collection.updateOne(<filter>, <update>, <options>)

db.collection.updateMany(<filter>, <update>, <options>)

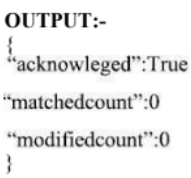
db.collection.replaceOne(<filter>, <update>, <options>)

SAMPLE QUERY:-updateOne()

INPUT:-

>>>db.inventory.updateOne({ item: "paper" }, { $set: { "size.uom": "cm"},
$currentDate: { lastModified: true } })

OUTPUT:-

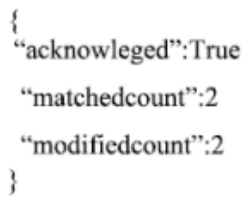


SAMPLE QUERY:-updateMany()

INPUT:-

>>>db.inventory.updateMany({ "qty": { $lt: 50 } }, { $set: { "size.uom": "in", status:
"P" }, $currentDate: { lastModified: true } })

OUTPUT:-



DELETE OPERATION:-

It is used to delete one or more documents/column from the collection based on the constraints.

Syntax:-

db.collection.deleteMany()

db.collection.deleteOne()

SAMPLE QUERY:-deleteOne()

INPUT:-

>>>db.inventory.deleteOne({ qty:85 })

OUTPUT:-

{
“acknowledged”:True
“deletecount”:1
}

SAMPLE QUERY:-deleteMany()

INPUT:-

>>>db.inventory.deleteMany({ qty:25 })

OUTPUT:-

{
“acknowledged”:True
“deletecount”:2
}

Indexing in MongoDB :

MongoDB uses indexing in order to make the query processing more efficient. If there is no indexing, then the MongoDB must scan every document in the collection and retrieve only those documents that match the query. Indexes are special data structures that store some information related to the documents such that it becomes easy for MongoDB to find the right data file. The indexes are ordered by the value of the field specified in the index.

Creating an Index :

MongoDB provides a method called createIndex() that allows users to create an index.

Syntax

db.COLLECTION_NAME.createIndex({KEY:1})

Example

db.mycol.createIndex({<age=:1})
{
<createdCollectionAutomatically= : false,
<numIndexesBefore= : 1,
<numIndexesAfter= : 2,
<ok= : 1
}

Dropping an Index :

In order to drop an index, MongoDB provides the dropIndex() method.

Syntax:

db.NAME_OF_COLLECTION.dropIndex({KEY:1})

The dropIndex() methods can only delete one index at a time. In order to delete (or drop) multiple indexes from the collection, MongoDB provides the dropIndexes() method that takes multiple indexes as its parameters.

Example

```
db.NAME_OF_COLLECTION.dropIndexes({KEY1:1, KEY2, 1})
```

Application

1. Web Applications

- MongoDB is widely used across various web applications as the primary data store.
- One of the most popular web development stacks, the MEAN stack employs MongoDB as the data store (MEAN stands for MongoDB, ExpressJS, AngularJS, and NodeJS).

2. Big Data

- MongoDB also provides the ability to handle big data.
- Big Data refers to massive data that is fast-changing, can be quickly accessed and highly available for addressing needs efficiently.
- So, it can be used in applications where Big Data is needed.

3. Demographic and Biometric Data

- MongoDB is one of the biggest biometrics databases in the world. It is used to store a massive amount of demographic and biometric data.
- For example, India's Unique Identification project, **Aadhar**, is using MongoDB as its database to store a massive amount of demographic and biometric data of more than 1.2 billion Indians.

4. Synchronization

- MongoDB can easily handle complicated things that need synchronization with each other entirely.
- So, it is mainly used in gaming applications. An example gaming application developed using MongoDB as a database is "EA".
- EA is a world-famous gaming studio that is using MongoDB Database for its game called FIFA Online 3.

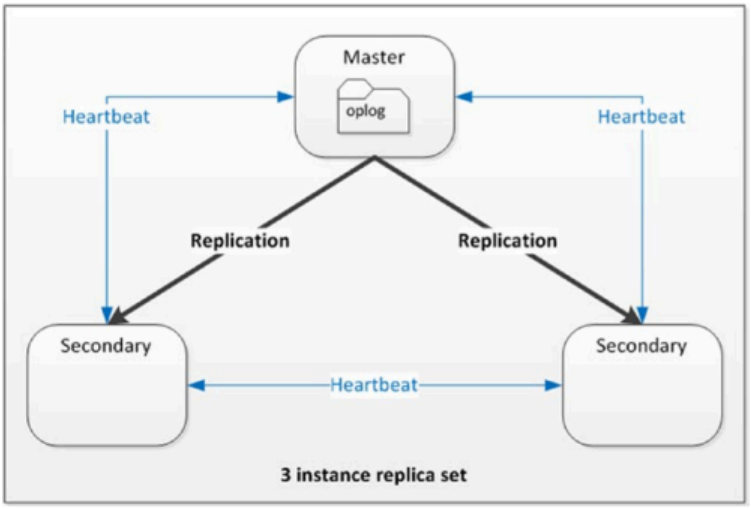
5. Ecommerce

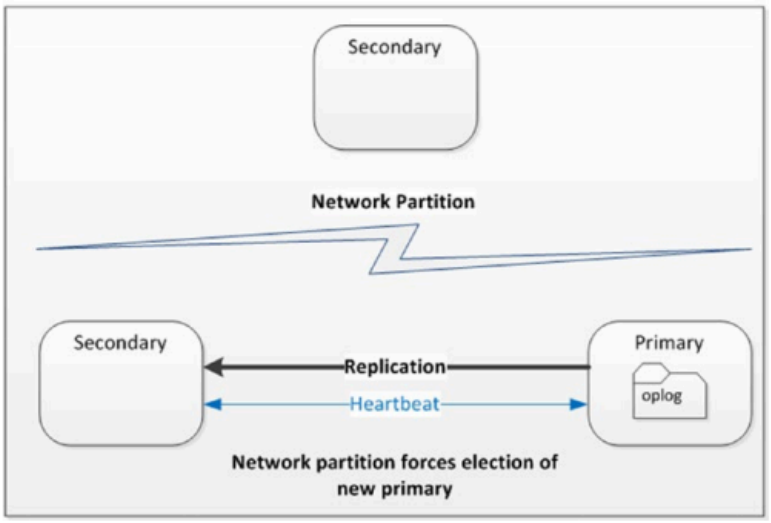
- For e-commerce websites and product data management and solutions, we can use MongoDB to store information because it has a flexible schema well suited for the job.
- They can also determine the pattern to handle interactions between user's shopping carts and inventory using "Inventory Management."

- MongoDB also has a report called “Category Hierarchy,” which will describe the techniques to do interaction with category hierarchies in MongoDB.

MongoDB Replication

- In MongoDB, data can be replicated across machines by the means of replica sets.
- A replica set consists of a primary node together with two or more secondary nodes.
- The primary node accepts all write requests, which are propagated asynchronously to the secondary nodes.
- The primary node is determined by an election involving all available nodes.
- To be eligible to become primary, a node must be able to contact more than half of the replica set.
- This ensures that if a network partitions a replica set in two, only one of the partitions will attempt to establish a primary.
- The successful primary will be elected based on the number of nodes to which it is in contact, together with a priority value that may be assigned by the system administrator.
- Setting a priority of 0 to an instance prevents it from ever being elected as primary.
- In the event of a tie, the server with the most recent optime — the timestamp of the last operation—will be selected.
- The primary stores information about document changes in a collection within its local database, called the oplog.
- The primary will continuously attempt to apply these changes to secondary instances. Members within a replica set communicate frequently via heartbeat messages.
- If a primary finds it is unable to receive heartbeat messages from more than half of the secondaries, then it will renounce its primary status and a new election will be called.
- Figure illustrates a three-member replica set and shows how a network partition leads to a change of primary.
- **Arbiters** are special servers that can vote in the primary election, but that don’t hold data.
- For large databases, these arbiters can avoid the necessity of creating unnecessary extra servers to ensure that a quorum is available when electing a primary.





The replication process works as follows:

- **Write operations on the primary:**
  - When a user sends a write operation (such as an insert, update, or delete) to the primary node, the primary node processes the operation and records it in its oplog (operations log).
- **Oplog replication to secondaries:**
  - Secondary nodes poll the primary's oplog at regular intervals.
  - The oplog contains a chronological record of all the write operations performed.
  - The secondary nodes read the oplog entries and apply the same operations to their data sets in the same order they were executed on the primary node.
- **Achieving data consistency:**

- Through this oplog-based replication, secondary nodes catch up with the primary's node data over time.
- This process ensures that the data on secondary nodes remains consistent with the primary's node data.

- **Read operations:**
  - While primary nodes handle write operations, both primary and secondary nodes can serve read operations which can help in load balancing.
  - Clients can choose to read from secondary nodes, which helps distribute the read load balance and reduce the primary node's workload.
  - But in some instances secondary nodes might have slightly outdated data due to replication lag.

MongoDB replication provides several benefits

- **High Availability:** In the event of primary node failure, a secondary node can be automatically promoted to the primary role, ensuring that the database remains operational and minimizing downtime.
- **Fault Tolerance:** Multiple replicas of data reduce the risk of data loss due to hardware failures or other issues affecting a single node.
- **Read Scalability:** Secondary nodes can handle read queries, distributing the read workload and improving overall performance.
- **Data Redundancy:** Having multiple replicas of data provides a level of data redundancy, helping protect against data loss.

In summary, MongoDB replication is a critical feature that enhances data availability and reliability in distributed environments. It enables the maintenance of synchronized data copies across multiple nodes, allowing for fault tolerance and improved performance in MongoDB database systems.

Methods to setup MongoDB replication

Setting up MongoDB replication is a crucial step in creating a fault-tolerant and highly available database environment. MongoDB replication allows you to create multiple copies of your data across different servers, ensuring data redundancy and fault tolerance. Here are the methods to set up MongoDB replication

MongoDB replication using replica set

Setting up MongoDB replication using a Replica Set involves several steps. Here are detailed instructions with code snippets for each step:

Step 1: Prepare MongoDB Instances

Install MongoDB on multiple servers or virtual machines where Replica Set will be created. You can follow the installation instructions provided in the MongoDB documentation.

Step 2: Configure Network Settings

Ensure that all the servers can communicate with each other over the network. Include the hostnames and IP addresses of all Replica Set members by updating them to the /etc/hosts file or DNS configuration.

Step 3: Start MongoDB Instances

For each server, a MongoDB configuration file needs to be created which will be saved as **mongod.conf** file name and written in **yaml** format type similar to the code snippet below.

```
storage:
  dbPath: /var/lib/mongodb
  journal:
    enabled: true
systemLog:
  destination: file
  path: /var/log/mongodb/mongod.log
  logAppend: true
net:
  bindIp:
  port:
replication:
  replSetName: myReplSet
```

To start MongoDB on each server using the configuration file made above i.e. mongod.conf file by using the following bash command:

```
mongod -f /path/to/mongod.conf
```

Step 4: Initialize the Replica Set

Now we need to connect to any one of the MongoDB instances (which is basically replica set) created using the bello MongoDB shell command:

```
mongo --host <hostname>:<port>
```

Replica Set needs to be initialized now by executing the following command:

```
rs.initiate(
    {_id: "myReplSet", members: [
    {_id: 0,host:"<primary_host>:<primary_port>"
```

}]
})

Step 5: Add Secondary Members

After executing the Replica Set, connect to the primary node using the following bash command in MongoDB shell:

mongo --host <primary_host>:<primary_port>

Add secondary members using the following Javascript command:

rs.add(":")

Repeat this step for each secondary member.

Step 6: Optional - Add Arbiter Node

If you want to add an arbiter node for elections, connect to the primary node's MongoDB shell and execute the following javascript command:

rs.addArb("<arbiter_host>:<arbiter_port>")

Step 7: Check Replica Set Status

To check the status of the Replica Set, connect to any of the MongoDB instances and run the following javascript command:

rs.status()

Step 8: Test Connection Failure

To test connection failure, you can simulate a primary node failure by stopping the MongoDB instance. The Replica Set should automatically elect a new primary node. Please note that the provided steps and code snippets are generalized and the actual steps might require adjustments based on the specific environment and use case. This is where a near real-time low code tool like fivetran can be leveraged as you just need to connect MongoDB with it and then Fivetran would handle all the replication tasks without any hassle.

While MongoDB replication using the replica set method offers numerous benefits, there are situations where its complexity, resource requirements, or alignment with specific use cases make it less feasible. Organizations need to carefully assess their requirements, infrastructure, and operational capabilities to determine whether replica sets are the appropriate solution or if alternative strategies should be considered.

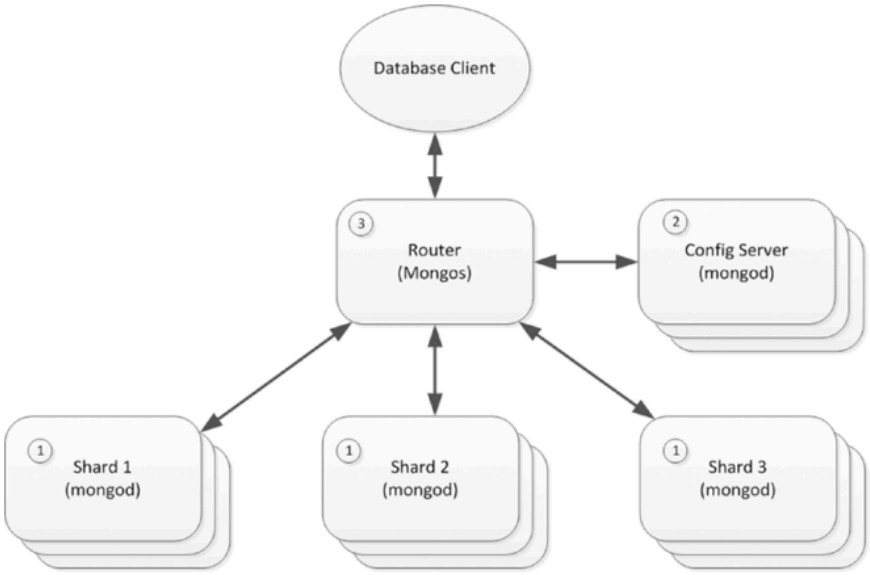
Sharding

A high-level representation of the MongoDB sharding architecture is shown in Figure. Each shard is implemented by a distinct MongoDB database, which in most respects is unaware of its role in the broader sharded server

(1) A separate MongoDB database

(2) the config server — contains the metadata that can be used to determine how data is distributed across shards.

(3) A router process — responsible for routing requests to the appropriate shard server.



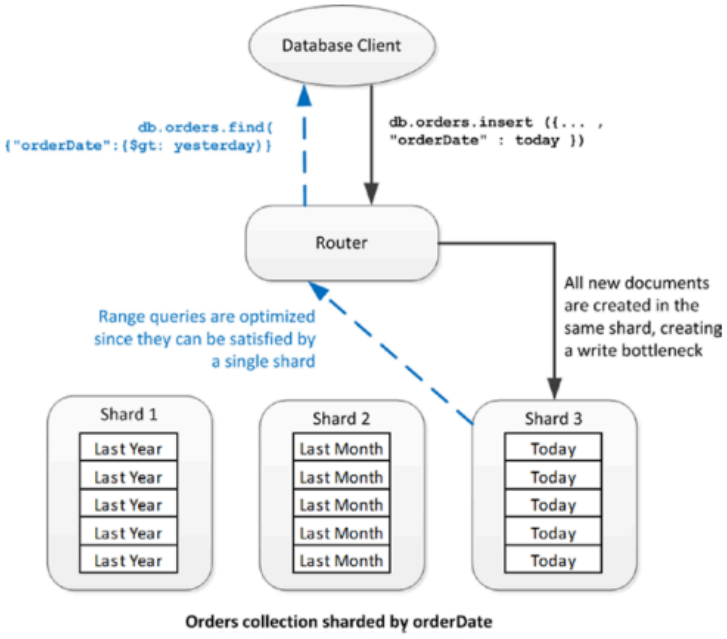
Sharding Mechanisms

Distribution of data across shards can be either **range based** or **hash based**.

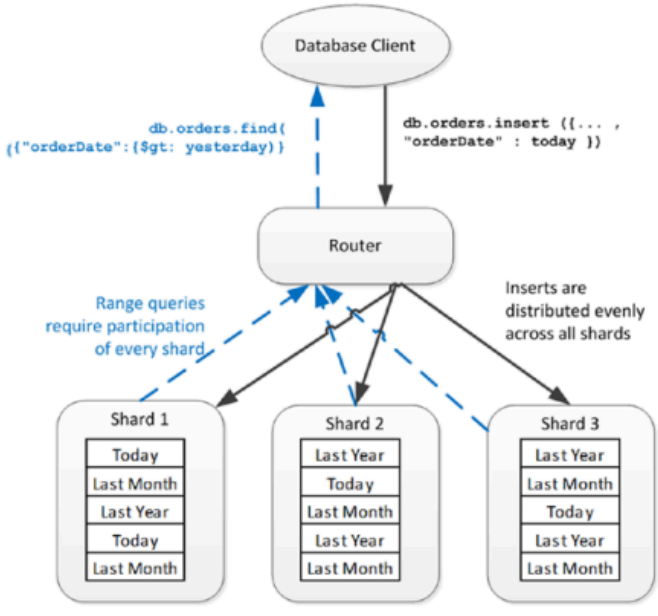
- **Range-based partitioning**:
  - Each shard is allocated a specific range of **shard key values**.
  - MongoDB consults the distribution of key values in the index to ensure that each shard is allocated approximately the same number of keys.
  - Range-based partitioning allows for more efficient execution of queries that process ranges of values, since these queries can often be resolved by accessing a single shard.
  - When range partitioning is enabled and the shard key is continuously incrementing, the load tends to aggregate against only one of the shards, thus unbalancing the cluster.
- **Hash-based sharding:**
  - The keys are distributed based on a **hash function** applied to the shard key.
  - Hash-based sharding requires that range queries be resolved by accessing all shards.

- Hash-based sharding is more likely to distribute documents (unfilled orders or recent posts) evenly across the cluster, thus **balancing the load** more effectively.
- With hash-based partitioning new documents are distributed evenly across all members of the cluster

The below Figure illustrates the performance trade-offs inherent in range and hash sharding for inserts and range queries.



Orders collection sharded by orderDate



Orders collection sharded by Hashed orderDate

- **Tag-aware sharding**:

- It allows the MongoDB administrator to fine-tune the distribution of documents to shards.
- By associating a shard with the tag, and associating a range of keys within a collection with the same tag, the administrator can explicitly determine the shard on which these documents will reside.
- This can be used to archive data to shards on cheaper, slower storage or to direct particular data to a specific data center or geography.
