Advanced Database Design Sharding

Gregory S DeLozier, Ph.D.

gdelozie@kent.edu

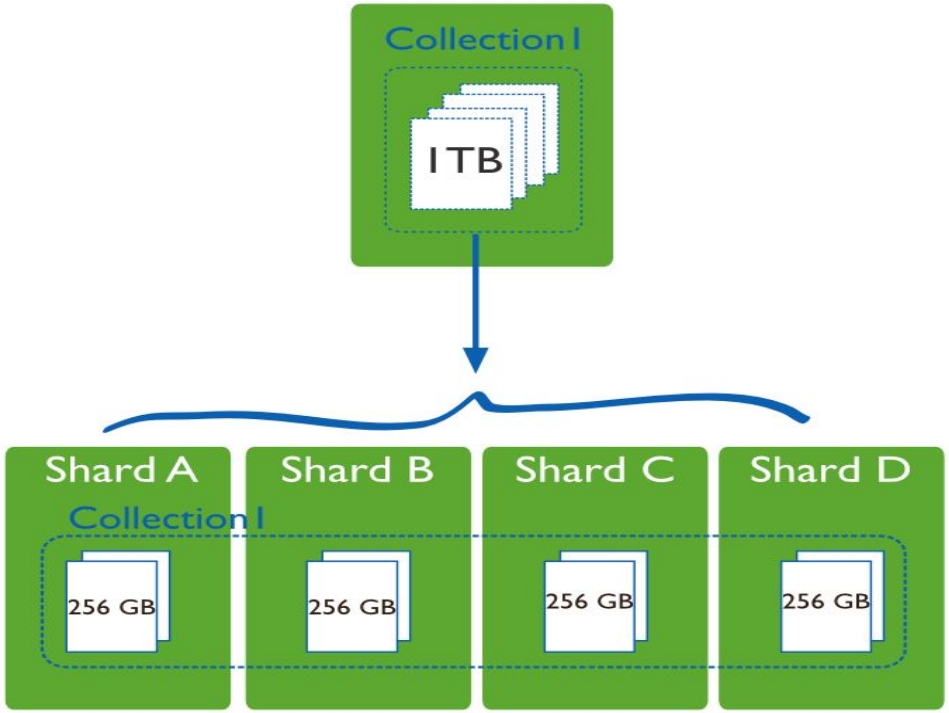
Sharding

- To increase *performance* of large databases
- Divide a collection into partitions called "shards" (how?)
- Put each partition onto a separate server
- Have master server that know what items are where
- All queries and commands go to the master server

- A server or database that supports this is "shard enabled"
- A collection partitioned this way is "sharded"

Sharding

Many shards combine
to create one collection.

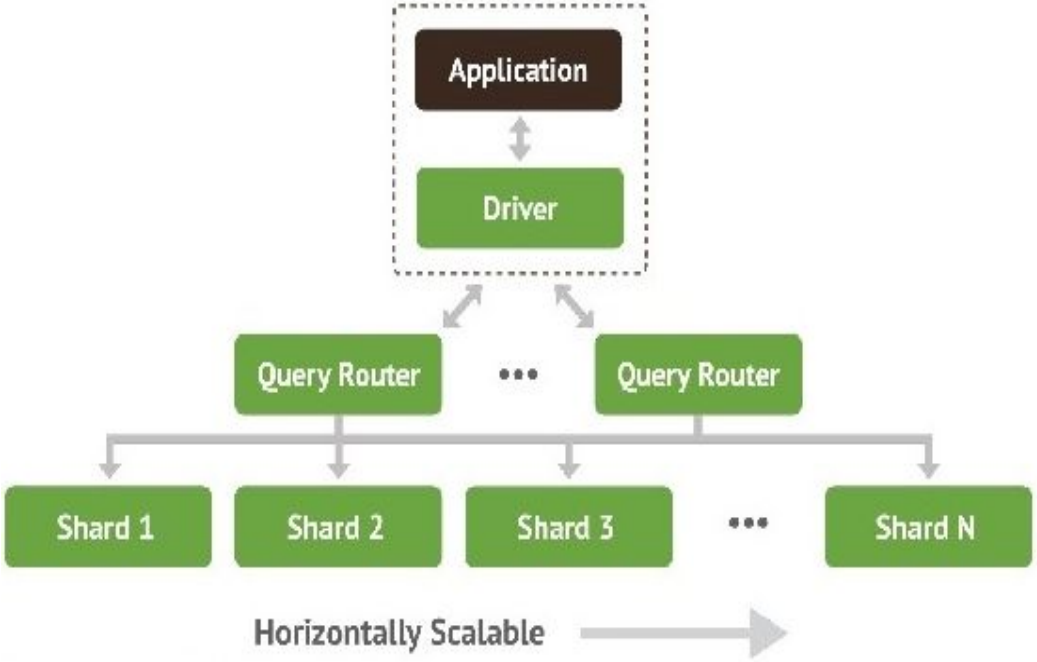


Query Routing

Queries are 'routed' to the shards responsible for containing the needed items.

This is transparent to the application.

Results may be gathered from many shards.

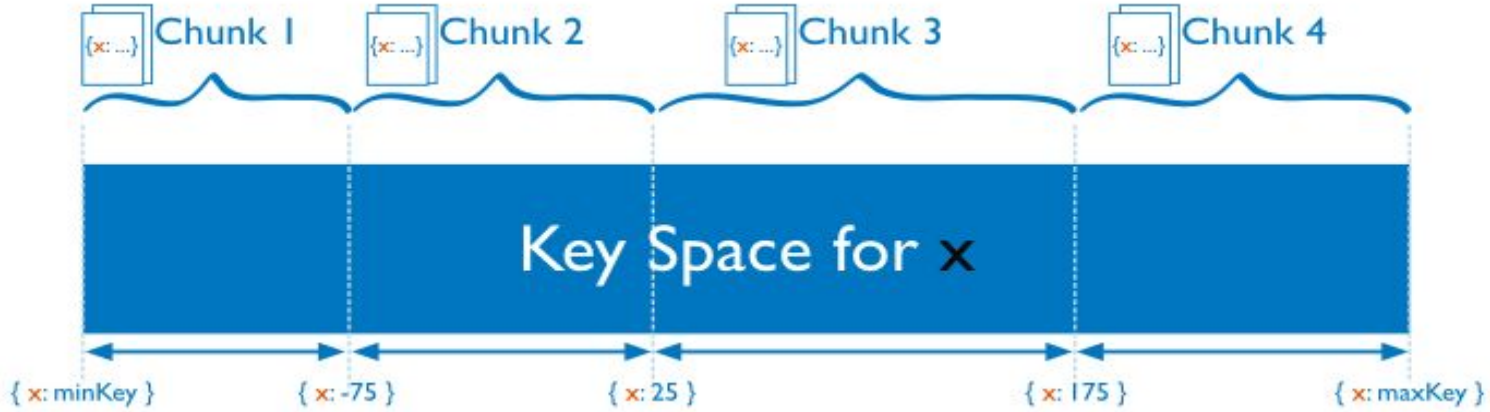


Sharding Keys

- Every collection that is sharded has a *sharding key*.
- This key decides what shard each item (or document) goes into.
- Keys that vary a lot tend to distribute the work across shards
    - This can be useful for broad queries
- Keys that are similar for similar queries tend to concentrate work to one shard
    - This can also be useful for single-case transactions

Shard Key Space

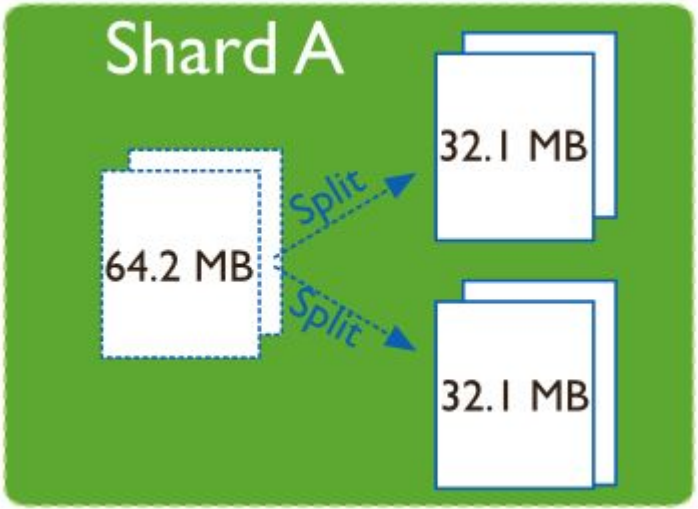
Key space is used to distribute ranges of the keys, or chunks, among shards.



When a chunk gets too large to handle in one shard, the chunk can be split, if the key allows. That's why having keys with lots of values is helpful.

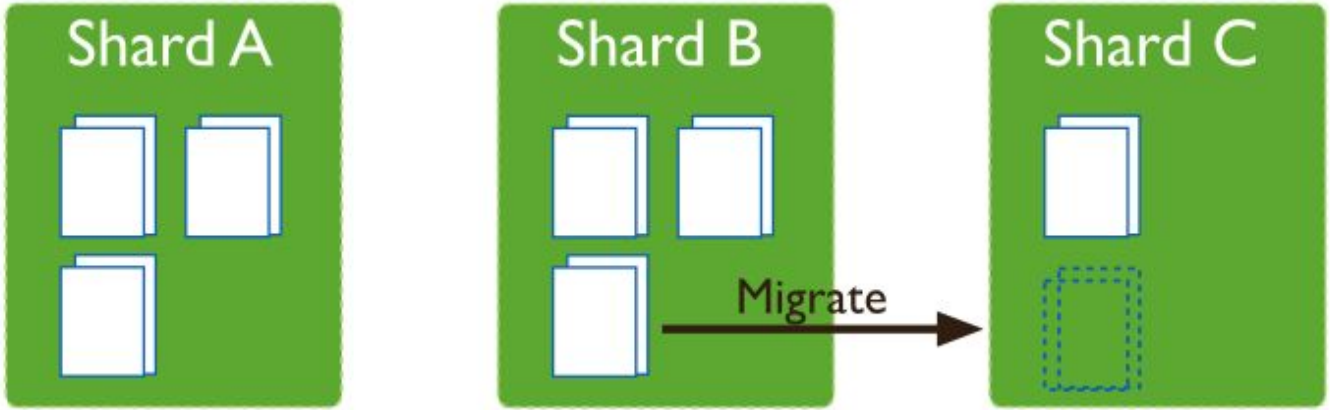
Splitting Chunks

Separating large chunks into smaller ones allows more effective use of shards.



Balancing Shards

After the chunk is split, one of the chunks is migrated (copied to a new shard and removed from the old one)

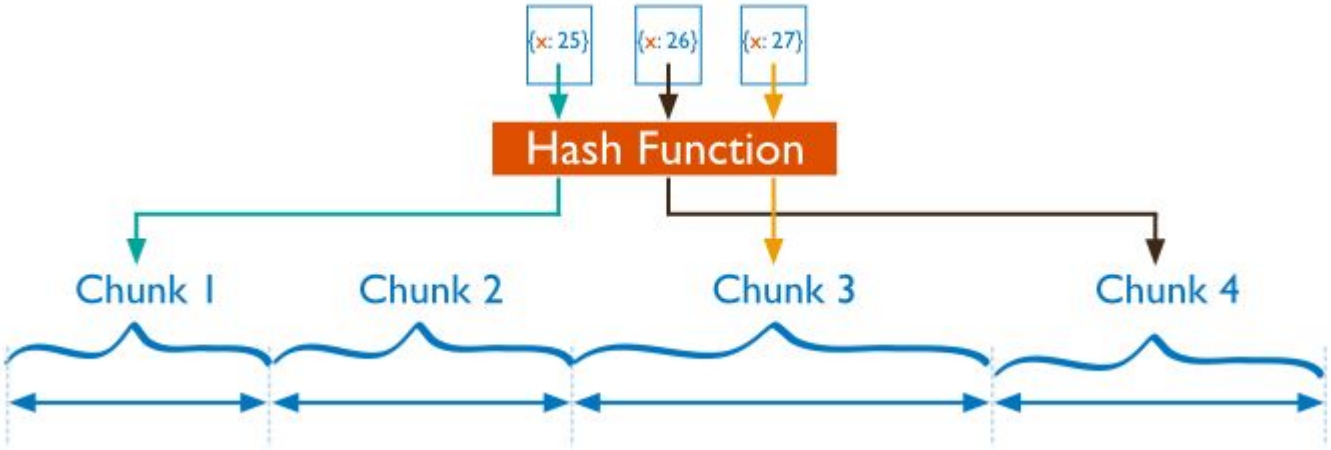


Choosing a Shard Key

- Having few values leads to difficulties with chunks
- Imagine a yes/no value. Only two possible chunks.
- Consider students at a high school.
  - Phone number -- probably a good key
  - Zip code -- probably not a good key -- few values
- Phone number would tend to associate one user's documents into one shard
- Dividing chunks would still keep related objects together
- At some point, you can just use a hash value as a sharding key
  - Random distribution
  - Not as big a problem where there aren't related sets of documents

Sharding Hash Function

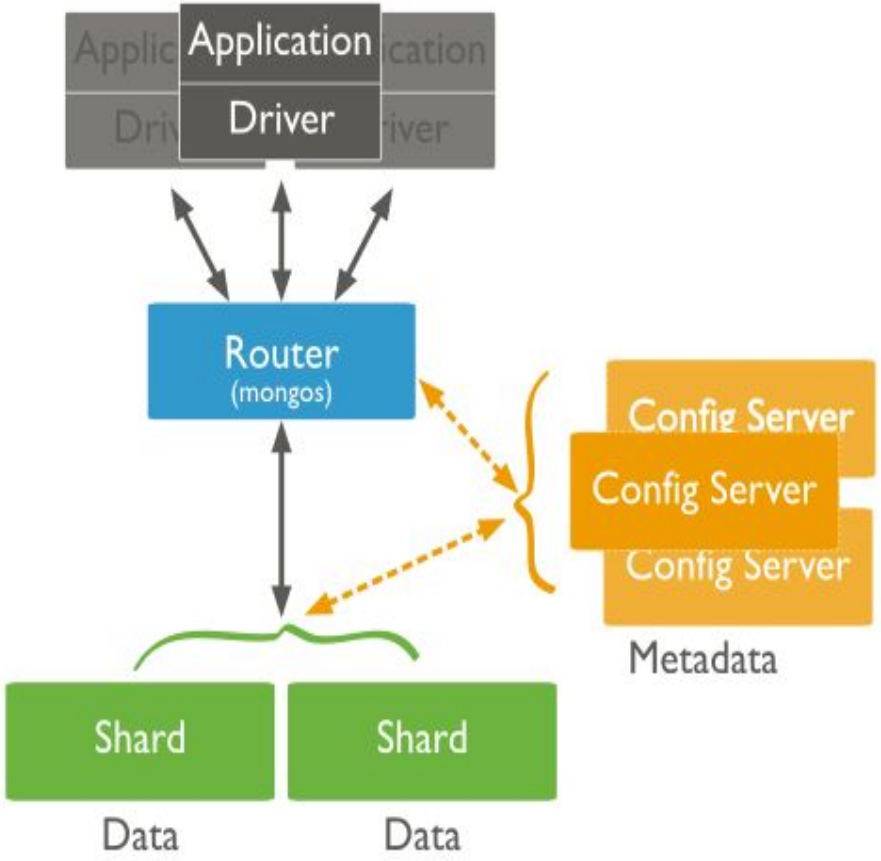
Hashes are automatically generated on the hash basis field.



Shard Routing with Keys

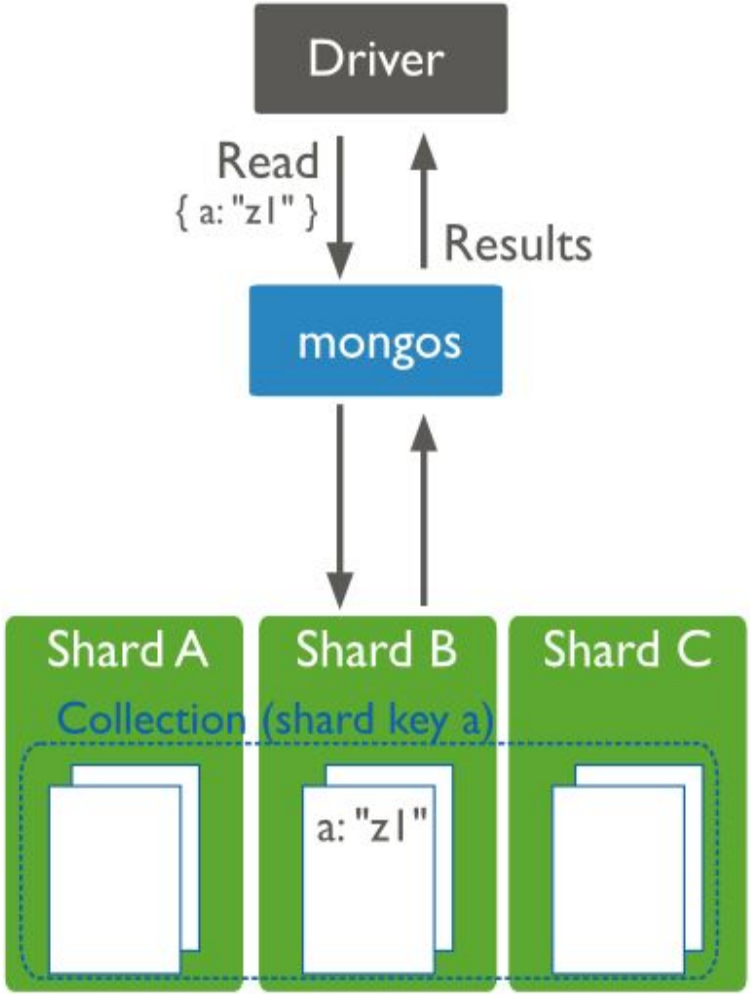
The router ("mongos") has to send queries to the shards.

Mongos uses metadata from Config servers to determine which shards to address, if the key is in the query.



Shard Routing with Keys

If the key is in the query, the router can determine the correct shard and only address one shard server.

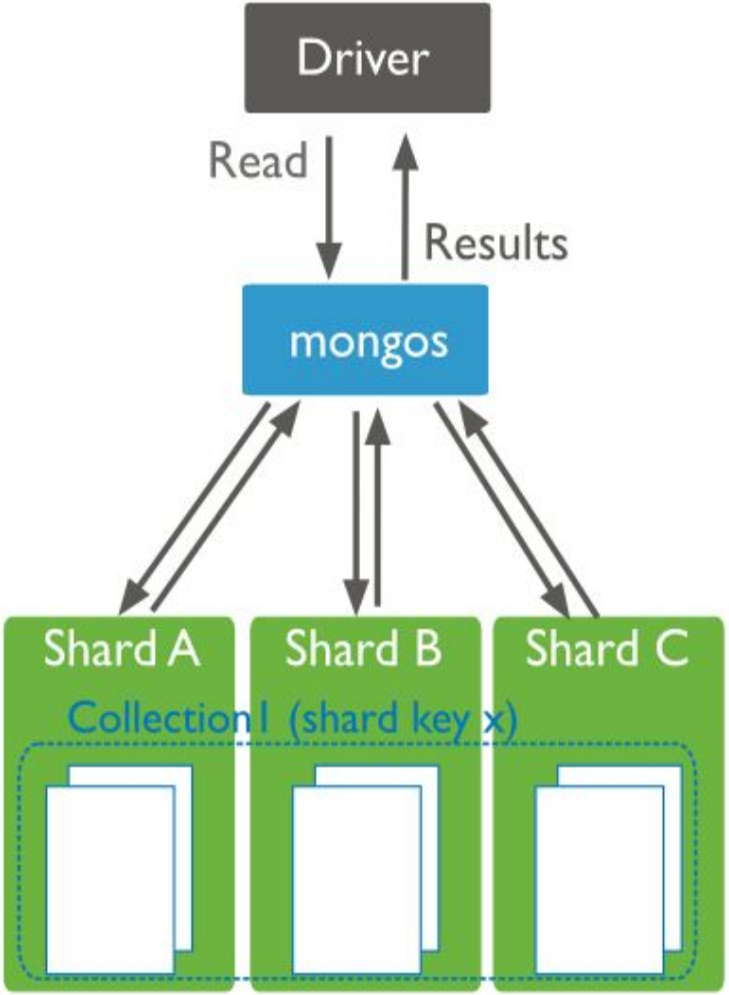


Shard Routing with Keys

If the key is not in the query, the router must query all shards in a "scatter query."

These are not efficient.

At very large scale, these may not be practical

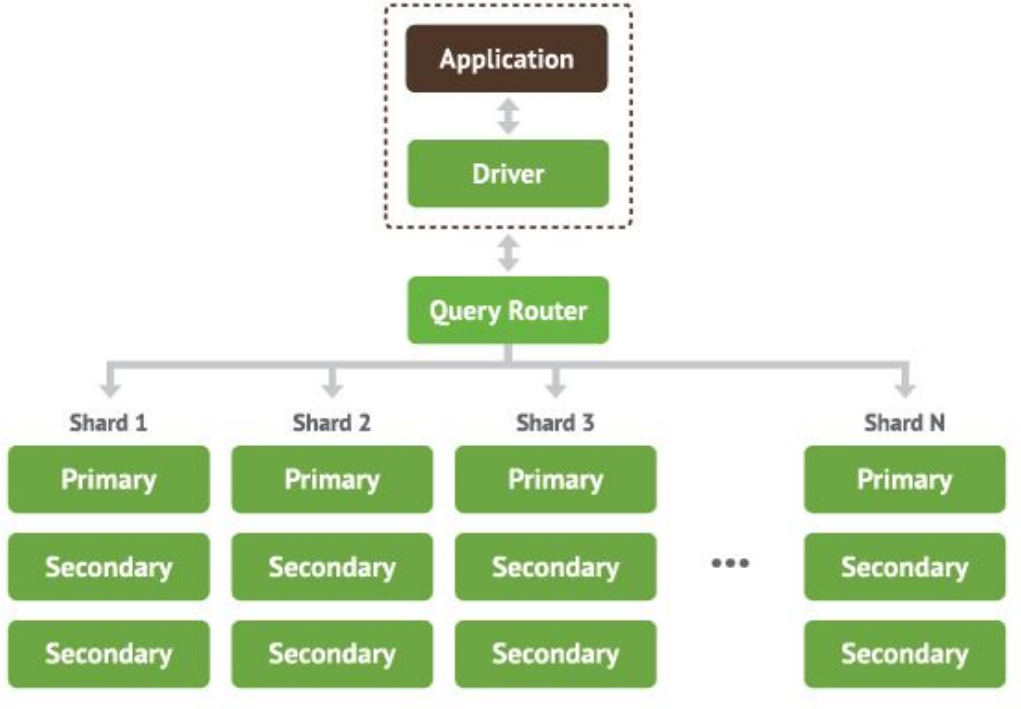


Replica Sets as Shards

Replica sets may be used as shards.

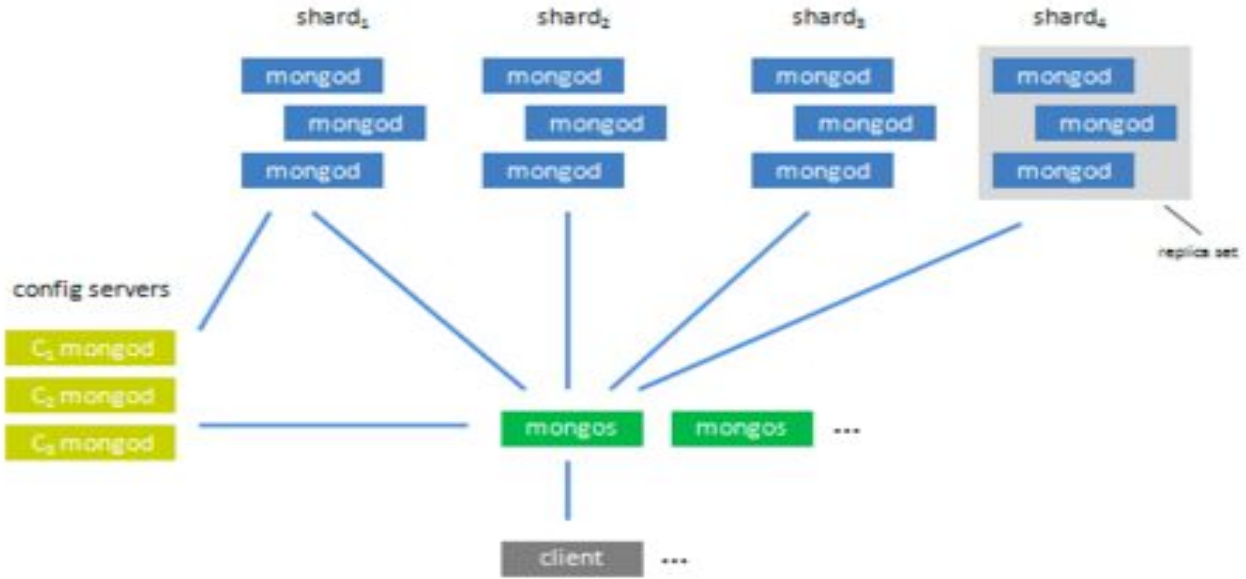
Replica sets are beneath the sharding layer.

Typically this is MxN servers, though various topologies are possible.



Many Servers to Set Up

18 servers here…



Mongo Considerations

- Flexible
  - Document Storage
  - Javascript in server
- Reliable
  - Replica sets
  - Auto fail-over
  - Add/remove replicas while live
- Capable
  - Sharding allows distribution of data and queries
  - Chunking allows balancing of shards
  - Possible to add shards and rebalance as needed

Deploying a Sharded Cluster

- Using one server for demo
- Create one config server instance
- Start *mongos* router using config server
- Start three mongo servers (mongod or replica sets)
- Connect to mongos via *mongo* client
- Add shards with *sh.addShard(...)*
- Enable sharding for a database
- Shard a collection (determine key, start sharding)
- Use the sharded collection

Start the 'configsvr' instance

- Normally three config servers, for demo we'll just use 1
  - Don't want the config data to be a bottleneck, but OK here

- Create a data directories for the config server
  - `$ mkdir -p ~/data/configdb`

- Start the server
  - `$ mongod --configsvr --dbpath ~/data/configdb --port 27011 &`

Start the Query Router

- Start the router
  - `$ mongos --configdb localhost:27011 --port 27021 &`

Create Some MongoDB Instances

- Create data directories for n = 1,2,3
  - `$ mkdir ~/data/db1`
  - `$ mkdir ~/data/db2`
  - `$ mkdir ~/data/db3`

- Start mongod servers for each shard for n = 1,2,3
  - `$ mongod --port 27031 --dbpath ~/data/db1`
  - `$ mongod --port 27032 --dbpath ~/data/db2`
  - `$ mongod --port 27033 --dbpath ~/data/db3`

Example Database

- Database: "students"
- Collection: "grades"
- Example document:

```
{
    "_id" : ObjectId("50906d7fa3c412bb040eb577"),
    "student_id" : 0,
    "type" : "exam",
    "score" : 54.132
}
```

Registering Shards

- $ mongo --port 27021 --host localhost

- mongos> sh.addShard("localhost:27031")
  { "shardAdded" : "shard0000", "ok" : 1 }

- mongos> sh.addShard("localhost:27032")
  { "shardAdded" : "shard0001", "ok" : 1 }

- mongos> sh.addShard("localhost:27033")
  { "shardAdded" : "shard0002", "ok" : 1 }

Enable Sharding on the Database

- mongos> sh.enableSharding("students")

  { "ok":1 }

Shard the Collection on a Key

- mongos> sh.shardCollection("students.grades", {"student_id": 1})

  { "collectionSharded" : "students.grades", "ok" : 1 }

Add Some Data

```
mongos> use students
switched to db students
mongos> for ( i = 200; i < 10000; i++ ) {
  db.grades.insert({student_id:i, type:"exam", score:Math.random()*100});
  db.grades.insert({student_id:i, type:"quiz", score:Math.random()*100});
  db.grades.insert({student_id:i, type:"home", score:Math.random()*100});
}
WriteResult({ "nInserted" : 1 })
```

Observe Sharding Activity

```
2015-11-23T21:54:08.110+0000 I SHARDING [conn1] moving chunk ns: students.grades
moving ( ns: students.grades, shard: shard0000:localhost:27031, lastmod:
1|3||000000000000000000000000, min: { student_id: 203.0 }, max: { student_id: MaxKey
}) shard0000:localhost:27031 -> shard0001:localhost:27032
```

Observe Chunk Transfer Progress

```
2015-11-23T21:54:16.721+0000 I SHARDING [conn1] moveChunk data transfer progress: {
active: true, ns: "students.grades", from: "localhost:27031", min: { student_id:
MinKey }, max: { student_id: 200.0 }, shardKeyPattern: { student_id: 1.0 }, state:
"ready", counts: { cloned: 0, clonedBytes: 0, catchup: 0, steady: 0 }, ok: 1.0 } my
mem used: 0

2015-11-23T21:54:16.723+0000 I SHARDING [conn1] moveChunk data transfer progress: {
active: true, ns: "students.grades", from: "localhost:27031", min: { student_id:
MinKey }, max: { student_id: 200.0 }, shardKeyPattern: { student_id: 1.0 }, state:
"ready", counts: { cloned: 0, clonedBytes: 0, catchup: 0, steady: 0 }, ok: 1.0 } my
mem used: 0
```

Observe Sharding Status

```
mongos> sh.status()
. . .
   students.grades
     shard key: { "student_id" : 1 }
     chunks:
       shard0000       1
       shard0001       1
       shard0002       1
     {"student_id":{"$minKey":1}}-->>{"student_id":200} on: shard0002 Timestamp(3,0)
     {"student_id":200}-->>{"student_id":203} on: shard0000 Timestamp(3,1)
     {"student_id":203}-->>{"student_id":{"$maxKey":1}} on: shard0001 Timestamp(2,0)

mongos>
```

Connect to Individual Shards

```
$ mongo --host localhost --port 27031

> use students

> db.grades.find().sort({student_id : 1}).limit(1)

> db.grades.find().sort({student_id : 1}).limit(1).pretty()

> db.grades.find().sort({student_id : -1}).limit(1).pretty()
```

Connect to Entire Cluster

```
$ mongo --host localhost --port 27021

> use students

> db.grades.find().sort({student_id : 1}).limit(1)

> db.grades.find().sort({student_id : 1}).limit(1).pretty()

> db.grades.find().sort({student_id : -1}).limit(1).pretty()
```

More Mongo Resources

https://docs.mongodb.org/getting-started/shell/

- MongoDB Shell (mongo)
- Python Edition
- Node.JS Edition
- C++ Edition
- Java Edition
- C# Edition

https://university.mongodb.com

http://www.javacodegeeks.com/2015/02/setting-up-sharded-mongodb-cluster-in-localhost.html

NoSQL Survey - review these pages

Survey - https://en.wikipedia.org/wiki/NoSQL

CouchDB - http://couchdb.apache.org/

BigTable - https://en.wikipedia.org/wiki/BigTable

Cassandra - https://en.wikipedia.org/wiki/Apache_Cassandra
http://cassandra.apache.org/

Redis - https://en.wikipedia.org/wiki/Redis
http://redis.io/