Replication and Sharding

Replication and Sharding

- So far we have run Mongo as a single server
- But Mongo was designed to run on a cluster of computers
  - Higher availability
  - Enable data replication across servers
  - Shard collections into many pieces
  - Perform queries in parallel

- Replication = duplication
  - Keeps identical copies running
  - Increase fault tolerance against the loss of a single database server
  - Makes sharding more robust

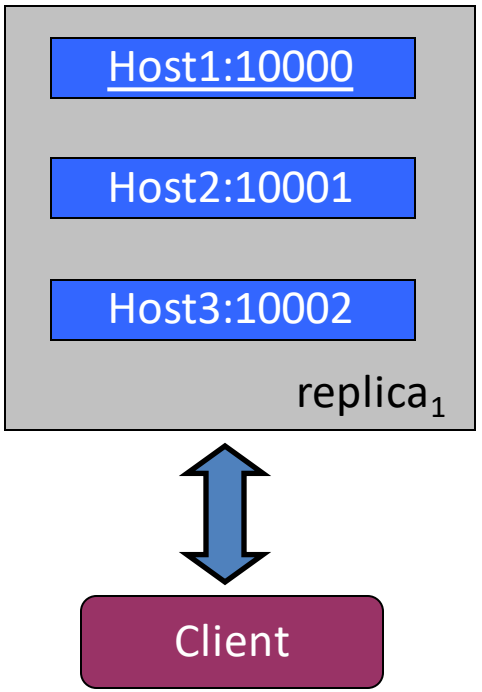
- Sharding distributes data across multiple machines

Replica Sets

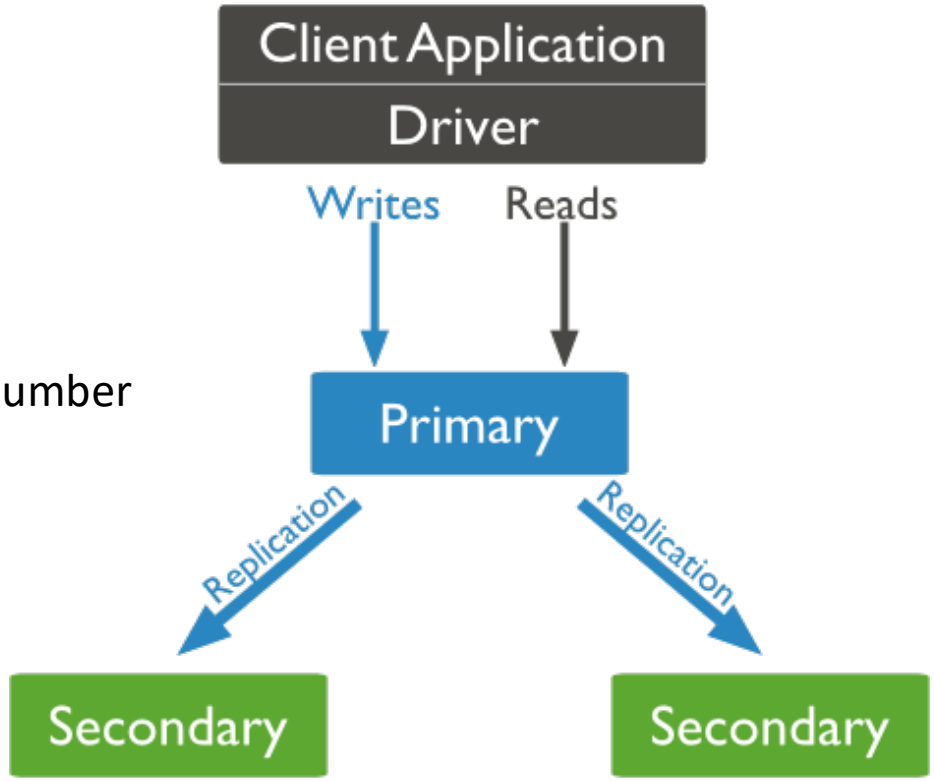
A replica set is a group of mongod instances that maintain the same data set.

Redundancy and Failover
Zero downtime for upgrades and maintaince

Master-slave replication
- Strong Consistency
- Delayed Consistency



Typical replication setup



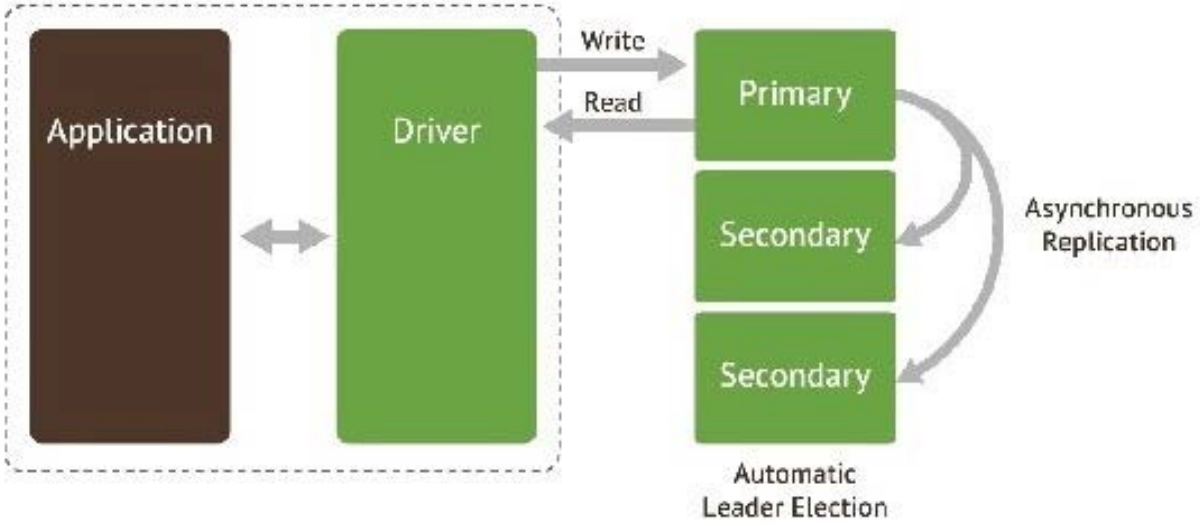
Clients normally talk to the Primary.

Usually an odd number works best!

What's the "majority" here?

Common sense says to put some of these at a different data center!

The replication is asynchronous…



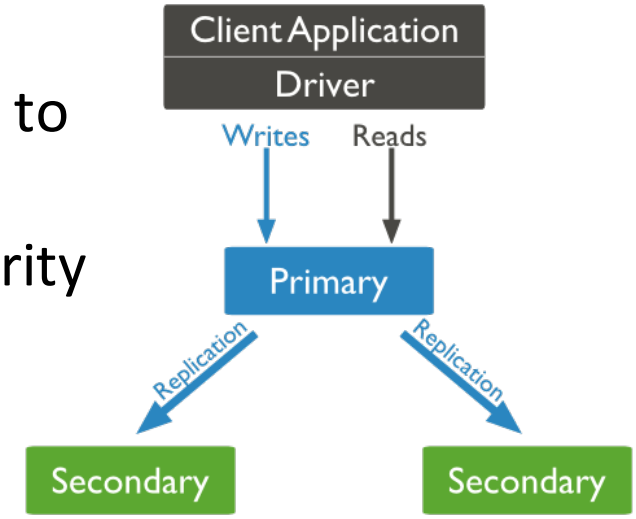
"Eventually up-to-date"

If a secondary goes down…
When it restarts, it will start synching from where it left off in its own oplog. May replay operations already applied – ok.

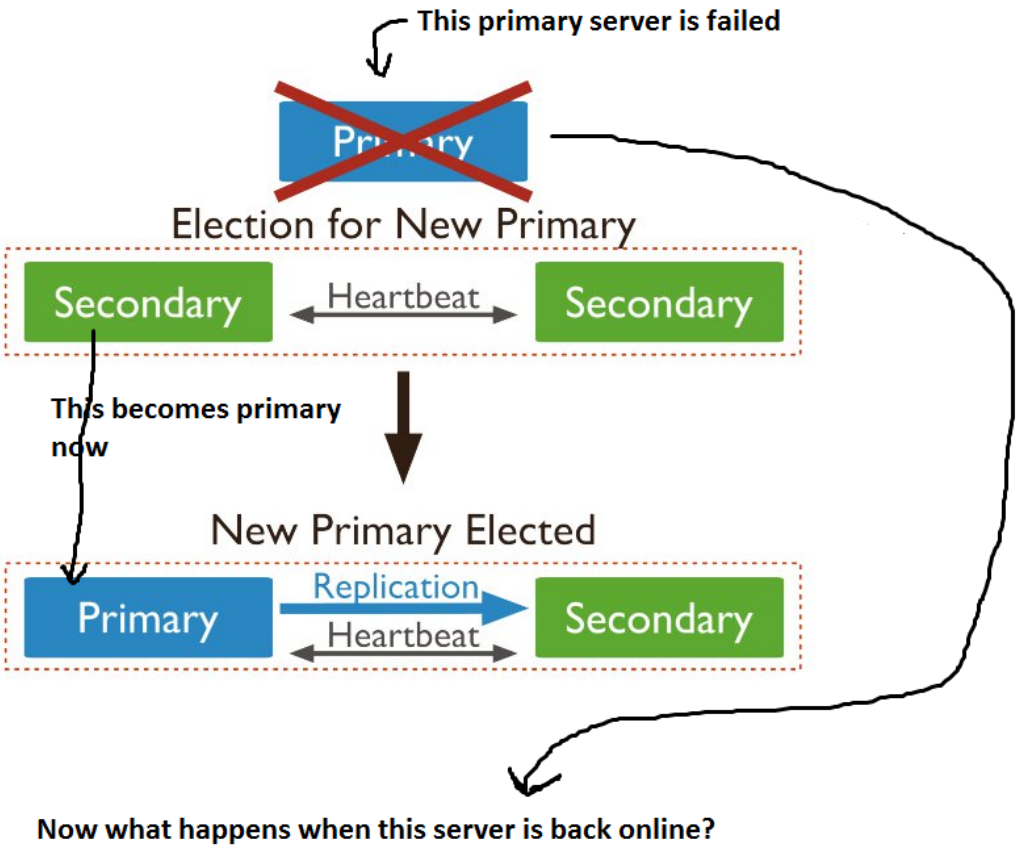
- If it's been down to long for this to work, it is "stale" and will attempt to make a full copy of the data from another member – initial synching.
  - Can also restore from backup, manually.
  - Recoveries can slow down operations.
    - E,g, "working set" in memory gets kicked out.
    - Need to rebuild indexes.

Replica set

- Heartbeats
  - Every two seconds, from every member to every other member.
  - Lets primary know if it can reach a majority of the set.
    - If not, it demotes itself!
  - Members communicate their state.
- Elections
  - If a member can't reach a primary, and is eligible to become a primary, it asks to become primary.
  - Other members go through logic to decide if this is suitable.



Rollbacks



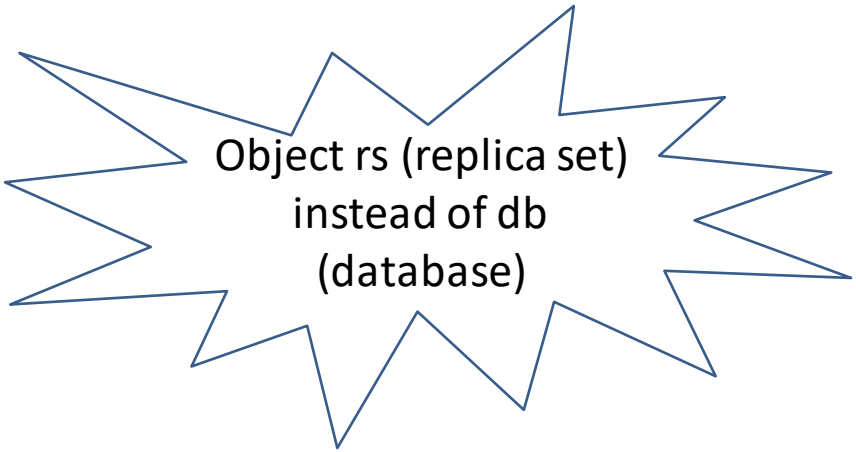
- Typically needed because of network partitions.
- Primary gets disconnected after a write, before replication.
- Can result in conflicting oplogs. Call the administrator!

Let's give it a try

- Today will start a few new servers. Mongo's default port is 27017, so we'll use other ports.
- Create three data directories
  - mkdir ./mongo1 ./mongo2 ./mongo3
- Next fire up the Mongo servers using the replSet flag
  - mongod --replSet moderndb --dbpath ./mongo1 --port 27011
  - mongod --replSet moderndb --dbpath ./mongo2 --port 27012
  - mongod --replSet moderndb --dbpath ./mongo2 --port 27013
- Now let's connect to the first server and initialize our replica set
  - mongo localhost:27011

```
> rs.initiate({id: 'moderndb',
    members: [
    {_id: 1, host: 'localhost:27011'},
    {_id: 2, host: 'localhost:27012'},
    {_id: 3, host: 'localhost:27013'}
    ]
    })

> rs.status().ok
```



Insert example

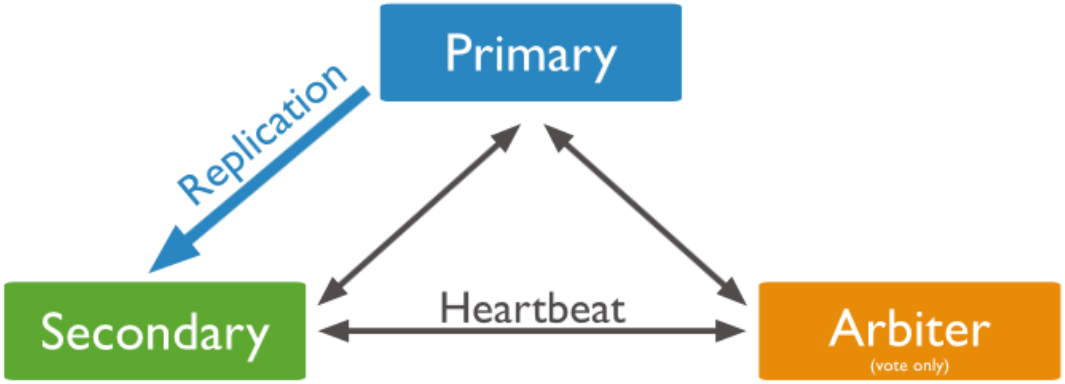
- Insert data in the client connected to the master server (state PRIMARY)
  - db.echo.insert({ say : *'HELLO!'* })
- Now exit the console and let's stop the PRIMARY server (Ctrl+C)
- Check on the console for the other two servers, one must have been promoted to master
- Connect to that server and search for the inserted value
  - db.echo.find()

- Now open a shell to the remaining secondary server and run the isMaster() function
  - db.isMaster().ismaster
  - db.isMaster().primary
- Let's try to insert another value
  - db.echo.insert({ say : *'can I insert here?'* })

- Now let's kill the current master
- And insert again, then restart the two servers

The problem with even number of nodes

- If network partition occurs, the majority of nodes that can still communicate would constitute the network
- The network without majority of nodes becomes no functional (the primary demotes itself to secondary)
- An arbiter is a voting but nonreplicating server in the replica set.

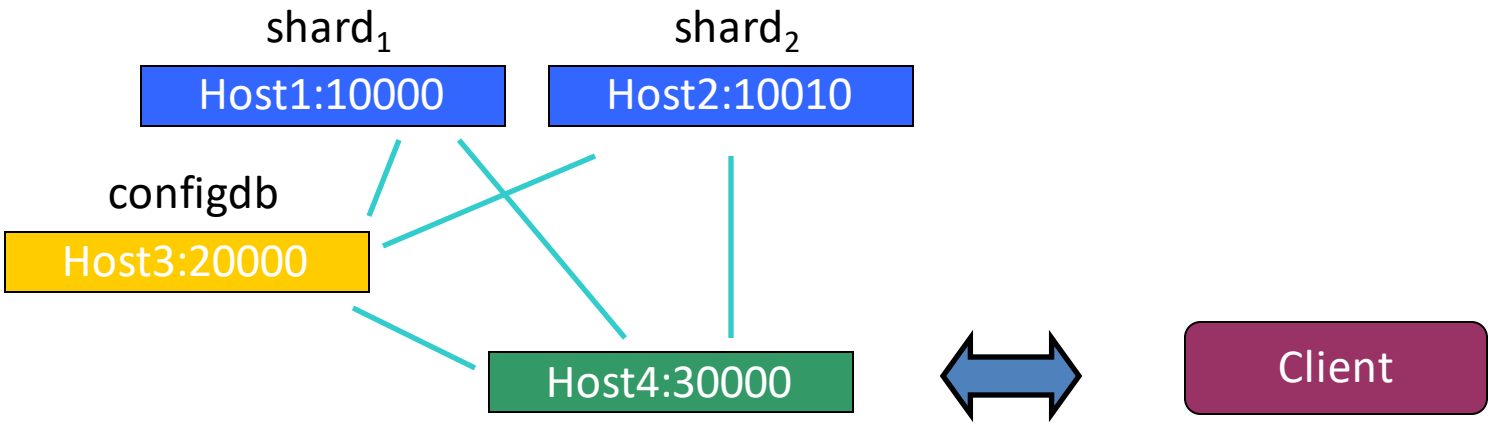


Sharding

id	company	customer	article	currency	price
4250250	020	073000	5994537812	00	142,50
4250251	020	073000	5994537852	00	141,12
4250252	020	073000	5994537854	00	105,99
4250253	020	073000	5994537856	00	109,52
4250254	020	073000	5994537862	00	131,49
4250255	020	073000	5994567308	00	29,86
4250256	020	073000	5994567422	00	57,13
4250257	020	073000	5994567428	00	68,59
4250258	020	073000	5994605089	00	51,09
4250259	020	073000	5994607975	00	93,93
4250260	020	073000	5994701005	00	74,22

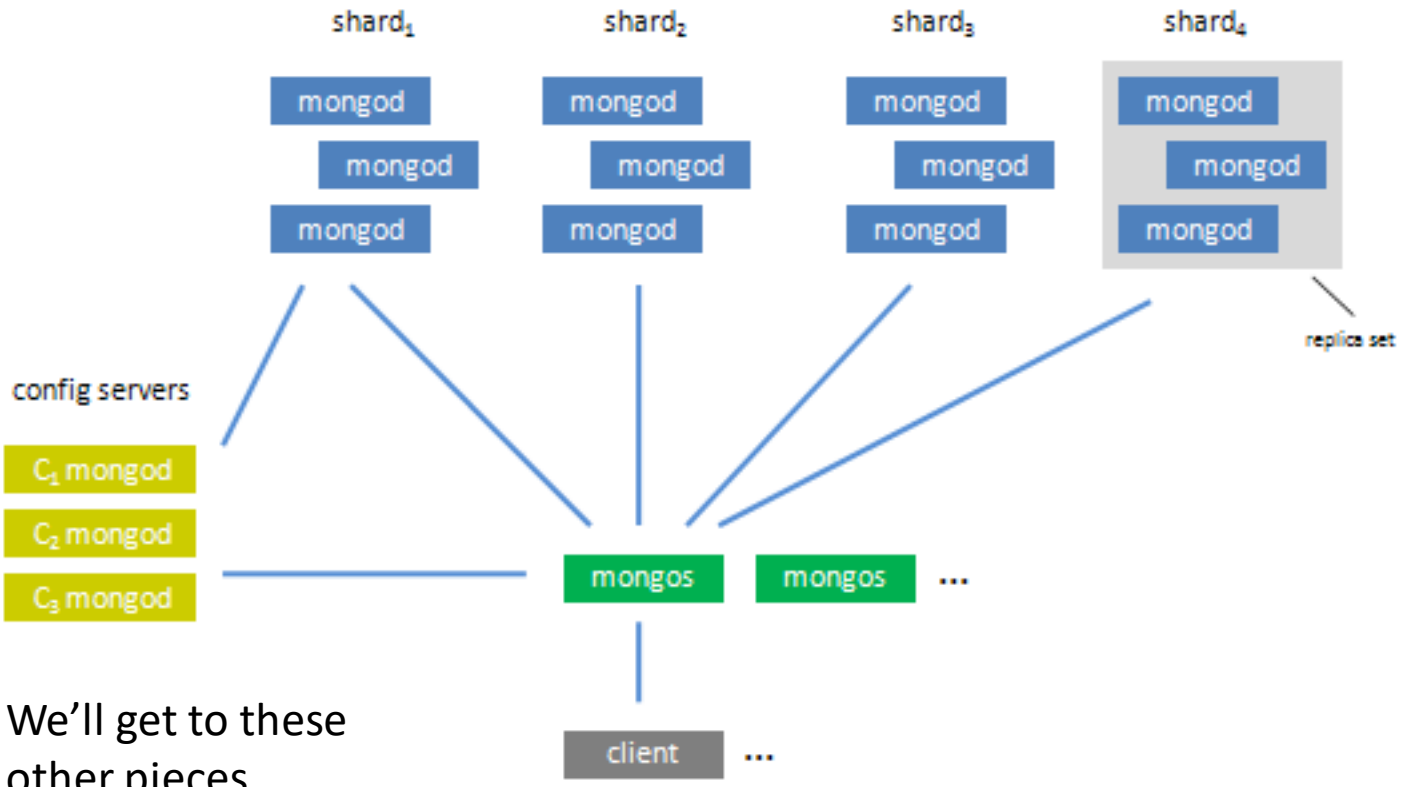
Partition your data
Scale write throughput
Increase capacity

Auto-balancing
MongoDB's built-in system abstracts the architecture and simplifies the admin.



Typically used *along with* replication

Don’t confuse with replication!

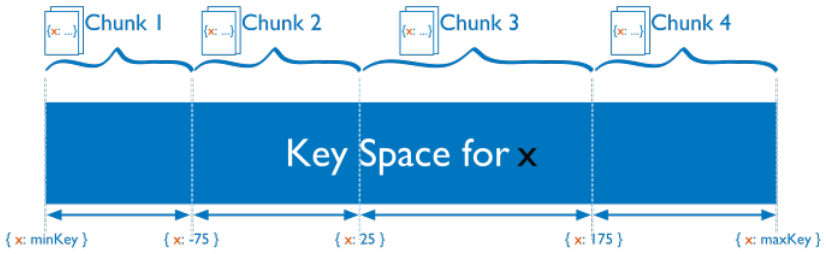


We’ll get to these other pieces.

Shard key

A field or two used to breakup the data.

- Like “username”.
- Must be indexed.
- MongoDB divides the data into “chunks” based on this key.
  - Each chunk is for a range of the keys.



The chunks are then evenly distributed across “shards.” (The separate servers being used.)

Client-side queries work normally. Routing done by “mongos”.

Can use “explain” to see what really happens.

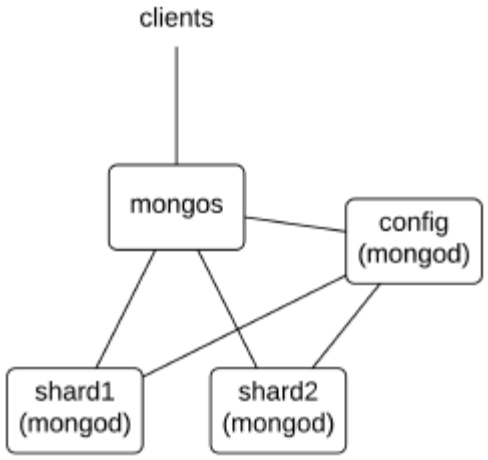
You can still do big operations on sharded datasets. e.g., mongos does sorting with a merge-sort across the shards.

Configuring Sharding

- When to shard
  - Increase available RAM.
  - Increase available disk space.
  - Reduce load on a server.
  - Read or write data with greater throughput than a single mongod can handle.
- Monitor to decide when sharding becomes necessary.
- Starting the servers
  - Need to set up the mongos and the shards.
  - Need to set up “config servers.” Usually 3!
    - These are “the brains of your cluster.” Used by mongos. “Table of contents.”
    - Set up first. Started first.
    - Each on a separate machine, geographically distributed.

Let's give it a try

- Let's launch a couple of (nonreplicating) mongod server. Parameter --shardsvr just means the server is capable of sharding
  $ mkdir ./mongo4 ./mongo5
  $ mongod --shardsvr --dbpath ./mongo4 --port 27014
  $ mongod --shardsvr --dbpath ./mongo5 --port 27015
- Now we need a config server to keep track of the keys
  $ mkdir ./mongoconfig
  $ mongod --configsvr --replSet configSet --dbpath ./mongoconfig --port 27016
- Let's configure the configSet replica set with only this server
  rs.initiate({ _id: *'configSet'*, configsvr: **true**, members: [{ _id: 0, host: *'localhost:27016'* }] })
- Now the mongos server will be entry point for our clients
  $ mongos --configdb configSet/localhost:27016 --port 27020
- Now let's connect to the mongos server admin database
  $ mongo localhost:27020/admin
- And configure some sharding
  - sh.addShard(*'localhost:27014'*)
  - sh.addShard(*'localhost:27015'*)



Sharding example, cont.

- Now we have to give it the database and collection to shard and the field to shard by
  - db.runCommand({ enablesharding : *"test"* })
  - db.runCommand({ shardcollection : *"test.zips"*, key : {city : 1} })
- Now let's import some data

```
$ mongoimport \
--host localhost:27020 \
--db test \
--collection zips \
--type json \
~\download\zips.json
```

- Now you can query the zips collection

```
mongo localhost:27020/test
```

  - db.zips.find({state: "IA"}).explain()

- Submit a screenshot of the result of this explain to ICON