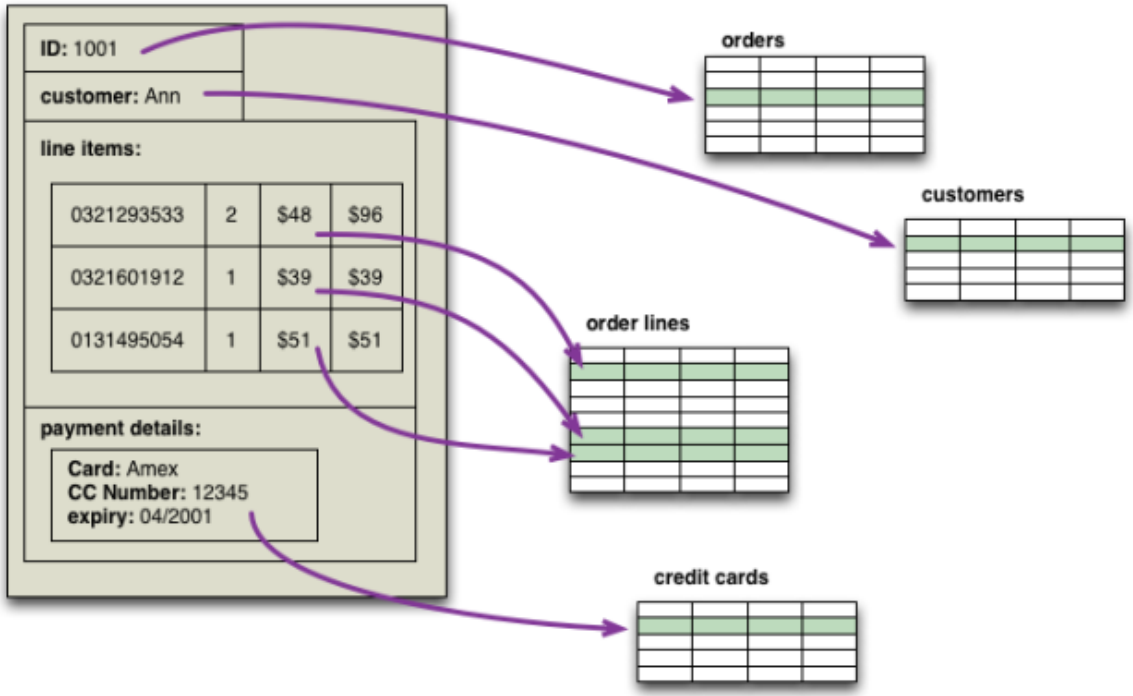
Document-base no SQL
The case of MongoDB

Marco Brambilla

Why document-based?

- Handles Schema Changes Well (easy development)
- Solves Impedance Mismatch problem
- Rise of JSON
  - python module: simplejson



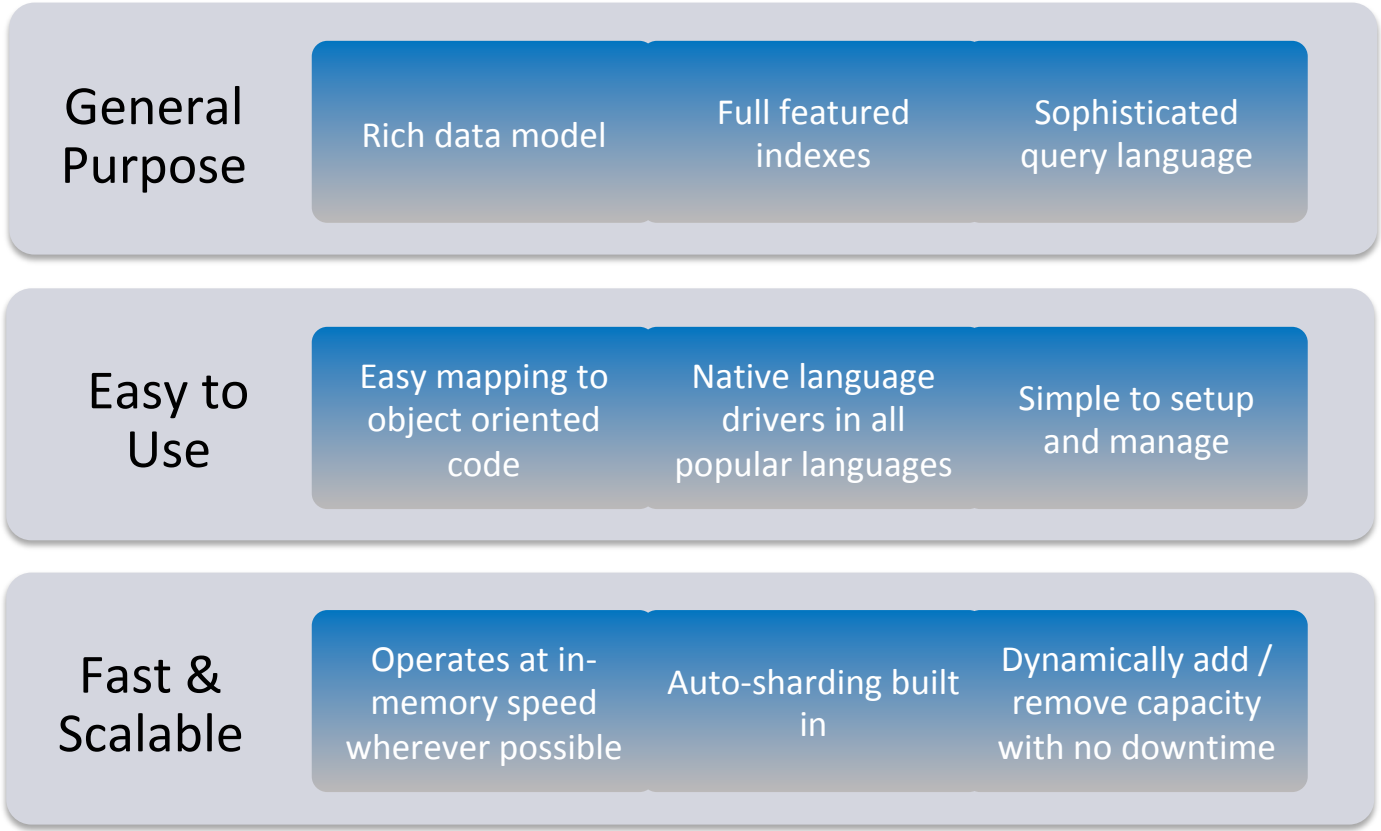
What is a document?


```
{
"business_id": "rncjoVoEFUJGCUoC1JgnUA",
 "full_address": "8466 W Peoria Ave\nSte 6\nPeoria, AZ 85345",
"open": true,
"categories": ["Accountants", "Professional Services", "Tax Services",],
 "city": "Peoria",
"review_count": 3,
"name": "Peoria Income Tax Service",
"neighborhoods": [],
"longitude": -112.241596,
"state": "AZ",
"stars": 5.0,
"latitude": 33.581867000000003,
"type": "business“
}
```

Overview – MongoDB

- An open source and document-oriented database.
- Data is stored in JSON-like documents.
- Designed with both scalability and developer agility.
- Dynamic schemas.
- Automatic data sharding

MongoDB is :


General Purpose
Rich data model
Full featured indexes
Sophisticated query language
Easy to Use
Easy mapping to object oriented code
Native language drivers in all popular languages
Simple to setup and manage
Fast & Scalable
Operates at in-memory speed wherever possible
Auto-sharding built in
Dynamically add / remove capacity with no downtime

Terminology: SQL vs MongoDB

SQL Terms/Concepts	MongoDB Terms/Concepts
database	database
table	collection
row	document
column	field
index	index
table joins (e.g. select queries)	embedded documents and linking
Primary keys	_id field is always the primary key
Aggregation (e.g. group by)	aggregation pipeline

Facts (1)

- No Schemas
- No transactions
- No joins
- Max document size of 16MB
  - Larger documents handled with GridFS

Facts (2)

- Runs on most common OSs
  - Windows
  - Linux
  - Mac
  - Solaris

- Data stored as BSON (Binary JSON)
  - used for speed
  - translation handled by language drivers

JSON Format

- Data is in name / value pairs
- A name/value pair consists of a field name followed by a colon, followed by a value:
  - Example: “name”: “R2-D2”
- Data is separated by commas
  - Example: “name”: “R2-D2”, race : “Droid”
- Curly braces hold objects
  - Example: {“name”: “R2-D2”, race : “Droid”, affiliation: “rebels”}
- An array is stored in brackets []
  - Example [{“name”: “R2-D2”, race : “Droid”, affiliation: “rebels”},
  - {“name”: “Yoda”, affiliation: “rebels”}]

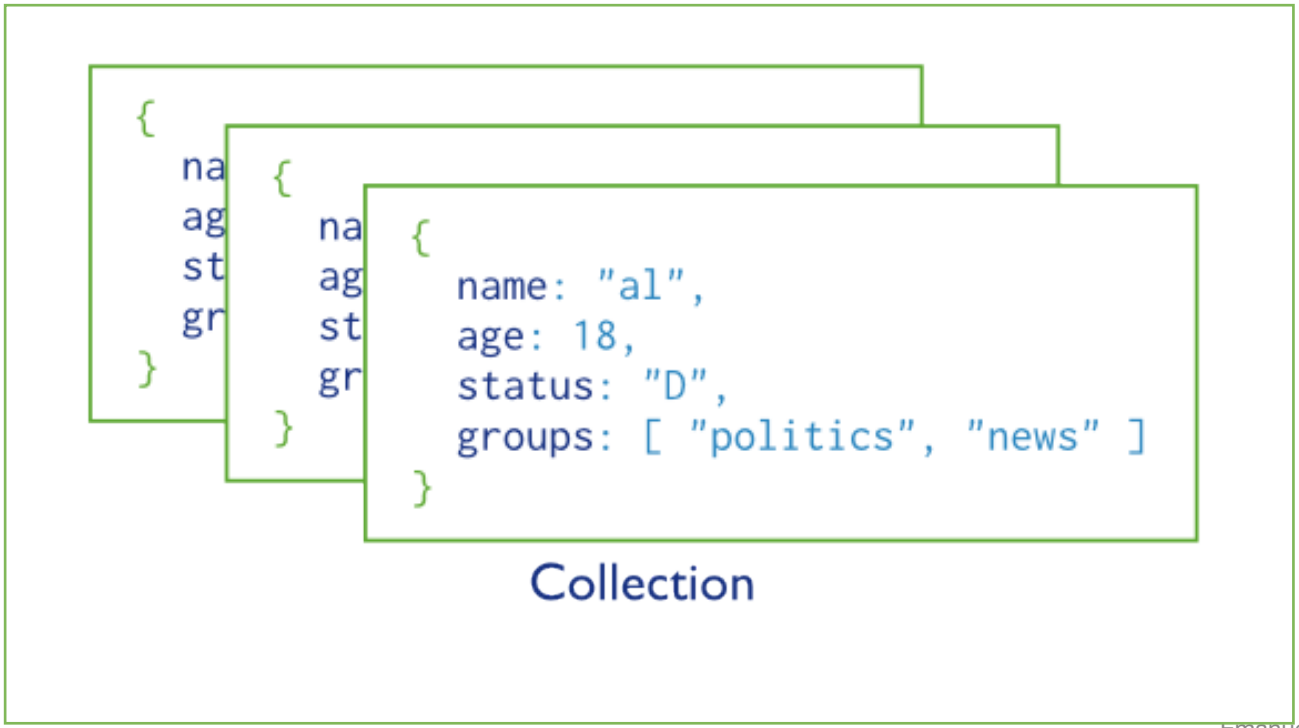
BSON Format

- Binary-encoded serialization of JSON-like documents
- Zero or more key/value pairs are stored as a single entity
- Each entry consists of a field name, a data type, and a value
- Files and large elements in a BSON document are prefixed with a length field to facilitate scanning
- Optimized for space and speed

BSON types are a superset of JSON types (JSON does not have a date or a byte array type, for example), with one exception of not having a universal "number" type as JSON does.

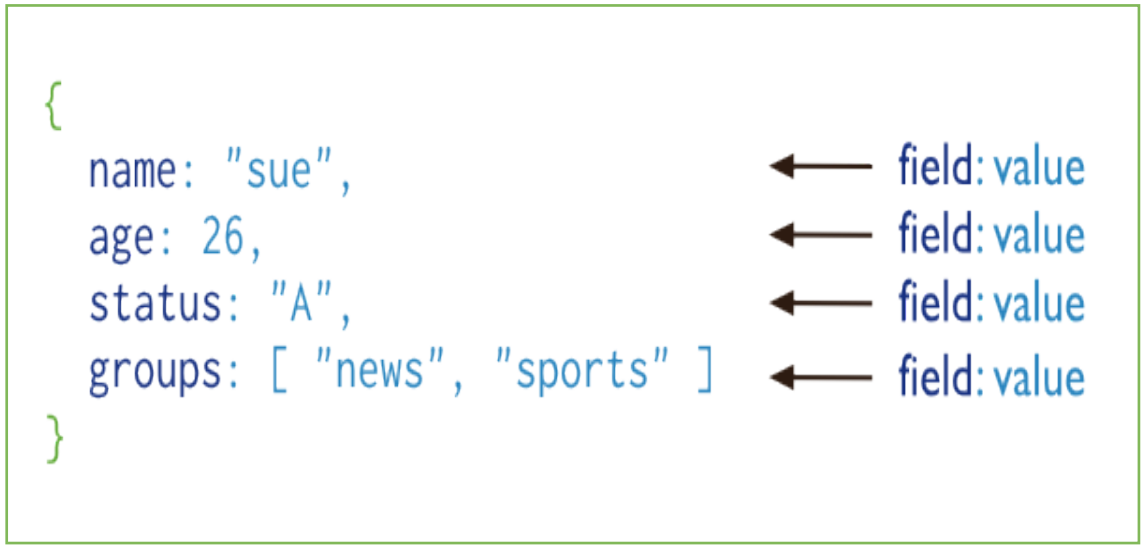
MongoDB Data Model

A ***collection*** includes ***documents***.



MongoDB Data Model

Structure of a JSON-document:



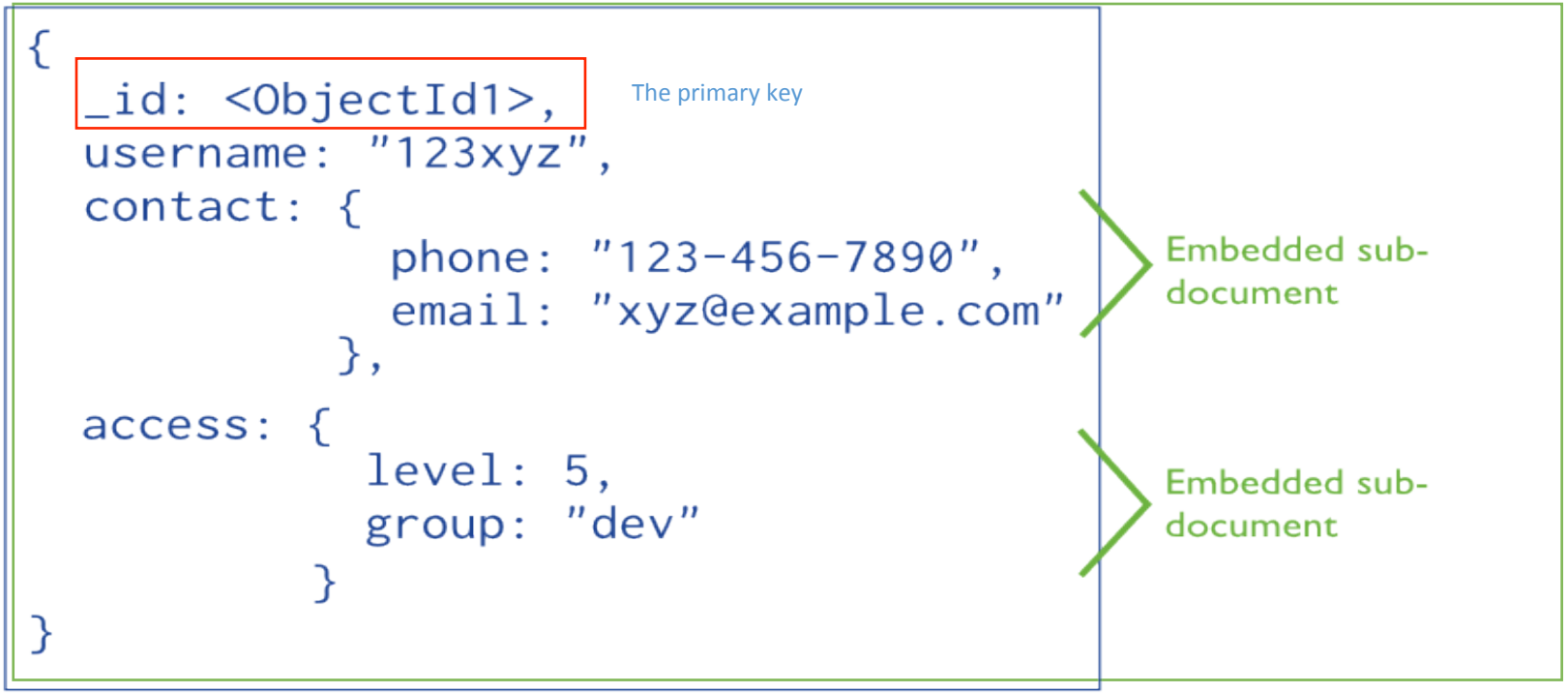
The value of **field**:

- Native data types
- Arrays
- Other documents

Rule: Every document must have an _id.

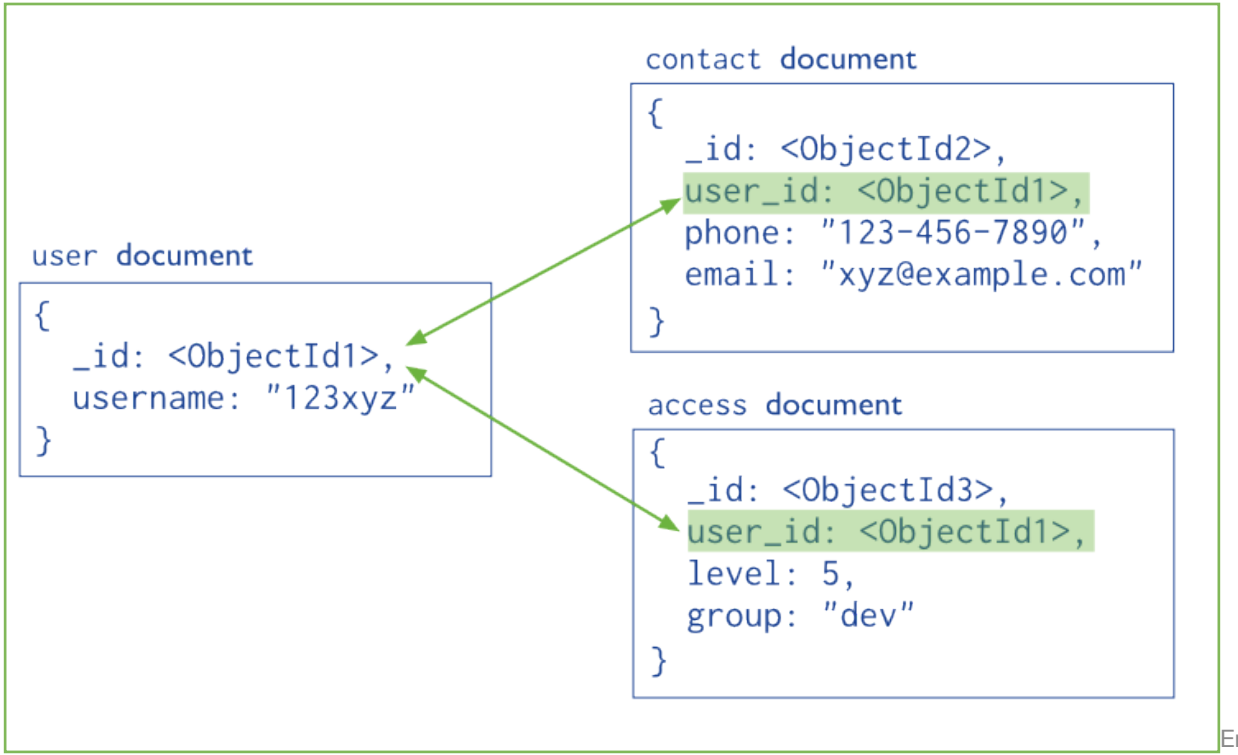
MongoDB Data Model

Embedded documents:



MongoDB Data Model

Reference documents or linking documents



MongoDB Queries: Create

CRUD (**Create** – Read – Update – Delete)

- Create a database: use database_name
- Create a collection:
  - db.createCollection(name, options)
    - options: specify the number of documents in a collection etc.
- Insert a document:
  - db.<collection_name>.insert({“name”: “nguyen”, “age”: 24, “gender”: “male”})

MongoDB Queries: Read

CRUD (Create – **Read** – Update – Delete)

- Query [e.g. select all]
  - db.<collection_name>.find().pretty()
- Query with conditions
  - db.<collection_name>.find(
    { “gender”: “female”, “age”: {$lte:20} }).pretty()
  - It’s pattern matching again!

Read – mapping to SQL

SQL Statement	MongoDB commands
SELECT * FROM table	db.collection.find()
SELECT * FROM table WHERE artist = ‘Nirvana’	db.collection.find({Artist:”Nirvana”})
SELECT* FROM table ORDER BY Title	db.collection.find().sort(Title:1)
DISTINCT	.distinct()
GROUP BY	.group()
>=, <	$gte, $lt

Comparison Operators

Name	Description
$eq	Matches value that are equal to a specified value
$gt, $gte	Matches values that are greater than (or equal to a specified value
$lt, $lte	Matches values less than or (equal to) a specified value
$ne	Matches values that are not equal to a specified value
$in	Matches any of the values specified in an array
$nin	Matches none of the values specified in an array
$or	Joins query clauses with a logical OR returns all
$and	Join query clauses with a loginal AND
$not	Inverts the effect of a query expression
$nor	Join query clauses with a logical NOR
$exists	Matches documents that have a specified field

https://docs.mongodb.org/manual/reference/operator/query/

Further Read Features: Aggregates

- SQL-like aggregation functionality
- Pipeline documents from a collection pass through an aggregation pipeline
- Expressions produce output documents based on calculations performed on input documents
- Example:

```
db.parts.aggregate ( {$group : {_id: type, totalquantity : { $sum: quanity} } } )
```

MongoDB Queries: Update

CRUD (Create – Read – **Update** – Delete)

- db.<collection_name>.update(<select_criteria>,<updated_data>)
- db.students.update({‘name':‘nguyen'}, { $set:{‘age': 20 } })
- Replace the existing document with new one: save method:
  - db.students.save({_id:ObjectId(‘string_id’),

“name”: “ben”, “age”: 23, “gender”: “male”}

MongoDB Queries: Delete

CRUD (Create – Read – Update – **Delete**)

- Drop a database
  - Show database: show dbs
  - Use a database: use <db_name>
  - Drop it: db.dropDatabase()
- Drop a collection:
  - db.<collection_name>.drop()
- Delete a document:
  - db.<collection_name>.remove({“gender”: “male” })

Processes

Mongod – Database instance

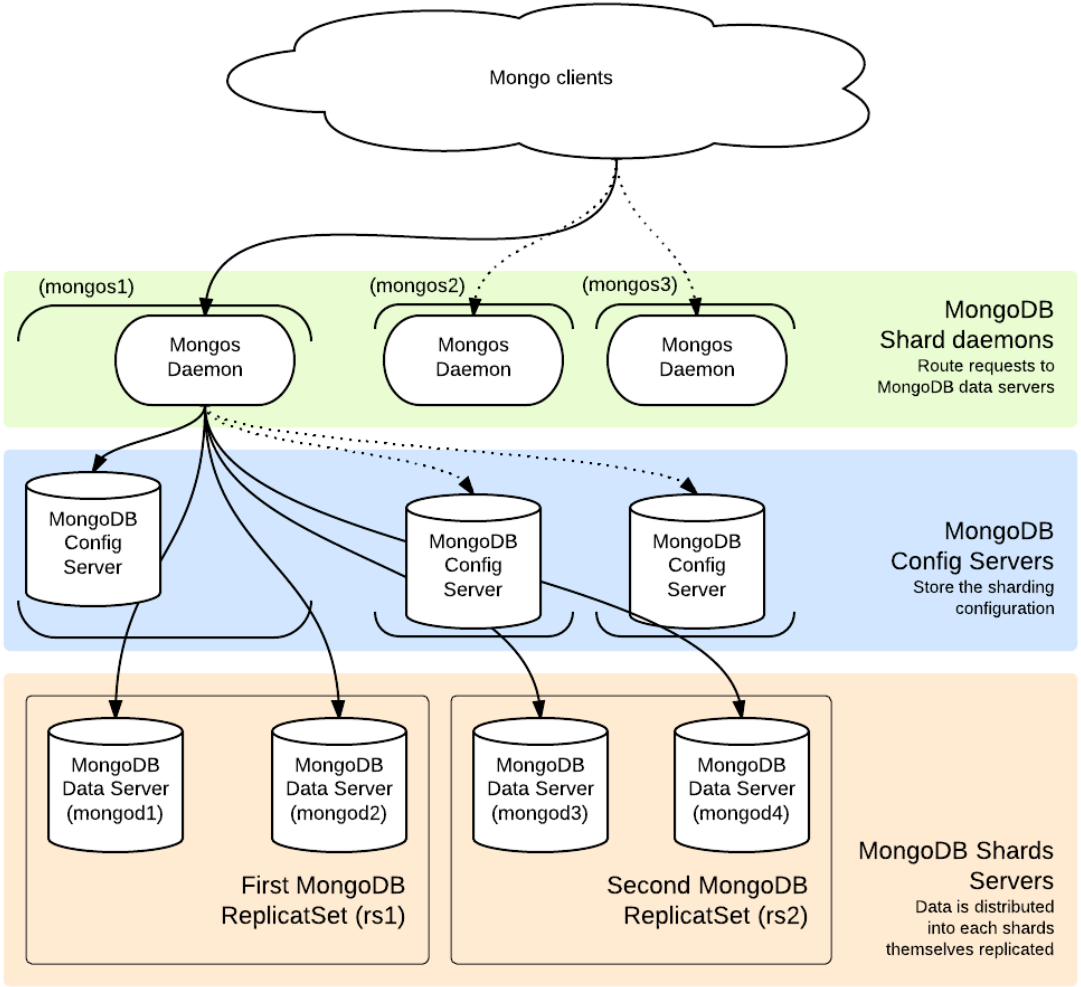
Mongos - Sharding processes

- Analogous to a database router.
- Processes all requests
- Decides how many and which mongod should receive the query
- Collates the results, and sends it back to the client.

Mongo – an interactive shell (a client)

- Fully functional JavaScript environment for use with a MongoDB

- You can have one mongos for the whole system no matter how many mongods you have
- OR you can have one local mongos for every client if you wanted to minimize network latency.



[source https://blog.sileht.net/using-a-shardingreplicaset-mongodb-with-ceilometer.html]

Indexes

- **B+ tree indexes**
- An index is automatically created on the _id field (the primary key)
- Users can create other indexes to improve query performance or to enforce Unique values for a particular field
- Supports single field index as well as Compound index
- Like SQL order of the fields in a compound index matters
- If you index a field that holds an array value, MongoDB creates separate index entries for every element of the array

Sparse Indexes

- **Sparse** property of an index ensures that the index only contain entries for documents that have the indexed field. (so ignore records that do not have the field defined)
- If an index is both unique and sparse – then the system will reject records that have a duplicate key value but allow records that do not have the indexed field defined

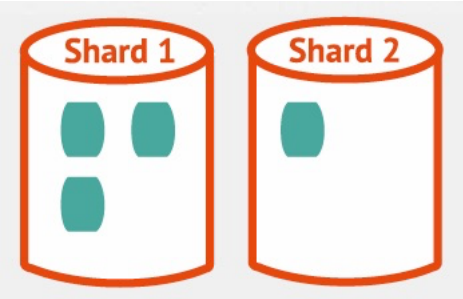
Sharding in MongoDB

- User defines shard key for partitioning
- Shard key defines range of data
  - Key space is like points on a line
  - Range is a segment of that line

Distribution

Initially 1 chunk

- Default max chunk size: 64mb
- MongoDB automatically splits & migrates chunks when max reached

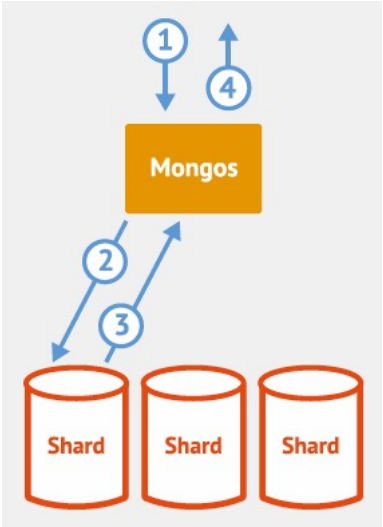


Routing and Balancing

• Queries routed to specific shards
• MongoDB balances cluster
• MongoDB migrates data to new nodes

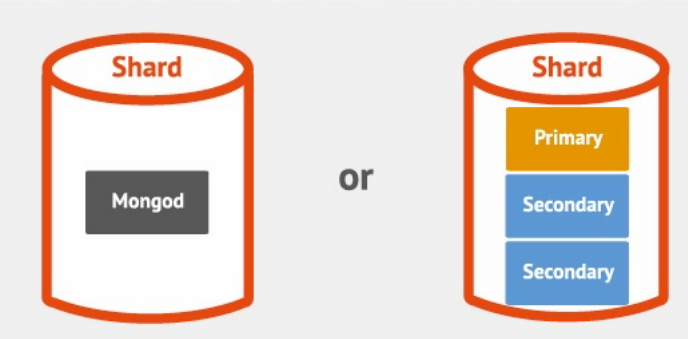
Config Server

– Stores cluster chunk ranges and locations
– Can have only 1 or 3 (production must have 3)



What is a Shard?

- Shard is a node of the cluster
- Shard can be a single mongod or a replica set



Auto-sharding

Minimal effort required

Two steps

– Enable Sharding for a database

– Shard collection within database

Mongod and Mongos

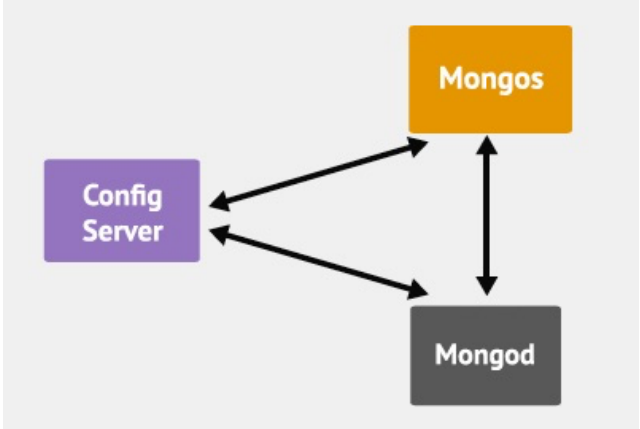
Mongod: main daemon

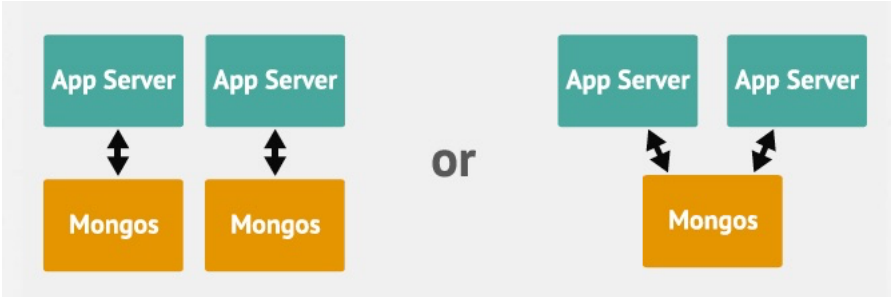
Mongos: Process that

– Acts as a router / balancer

– No local data (persists to config database)

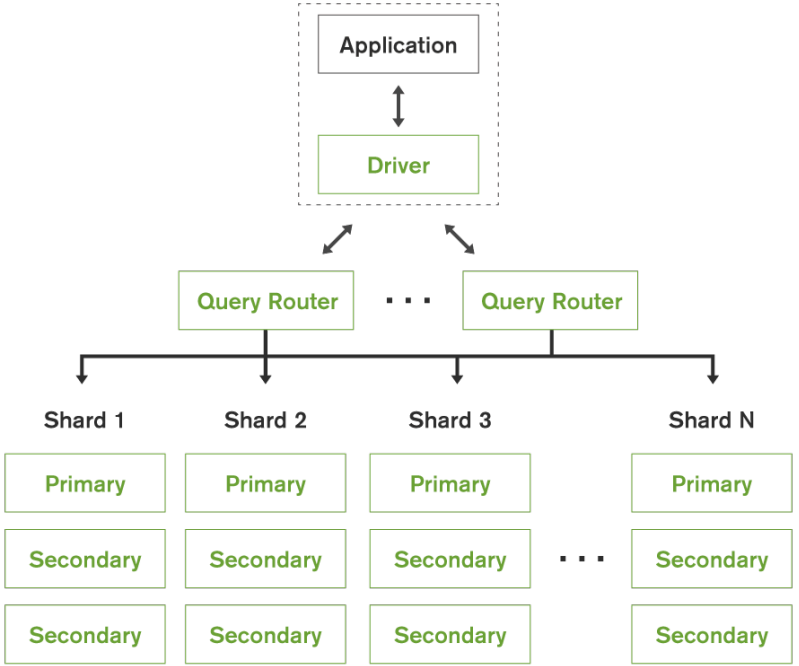
– Can have 1 or many





MongoDB Sharding Strategies

- Ranged
- Hashed
- Tag-aware



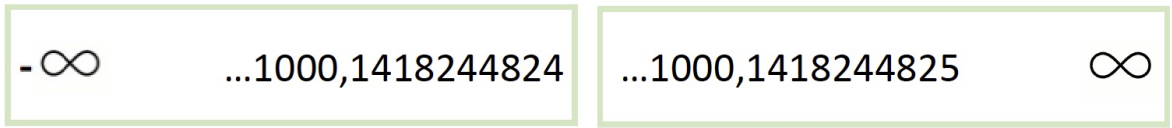
Range Sharding

Splits shards based on sub-range of a key

(or also multiple keys combined)

Simple Shard Key: {deviceId}

Composite Shard Key: {deviceId, timestamp}



Hash Sharding

A subset of Range Sharding.

MongoDB apples a MD5 hash on the key when a hash shard key is used:

```
Hash Shard Key(deviceId) = MD5(deviceId)
```

Ensures data is distributed randomly within the range of MD5 values

Tag Sharding

Tag-aware sharding allows subset of shards to be tagged, and assigned to a sub-range of the shard-key.

Example: Sharding User Data belong to users from 100 “regions”

Collection: Users, Shard Key: {uId, regionCode}

Tag based on macro regions

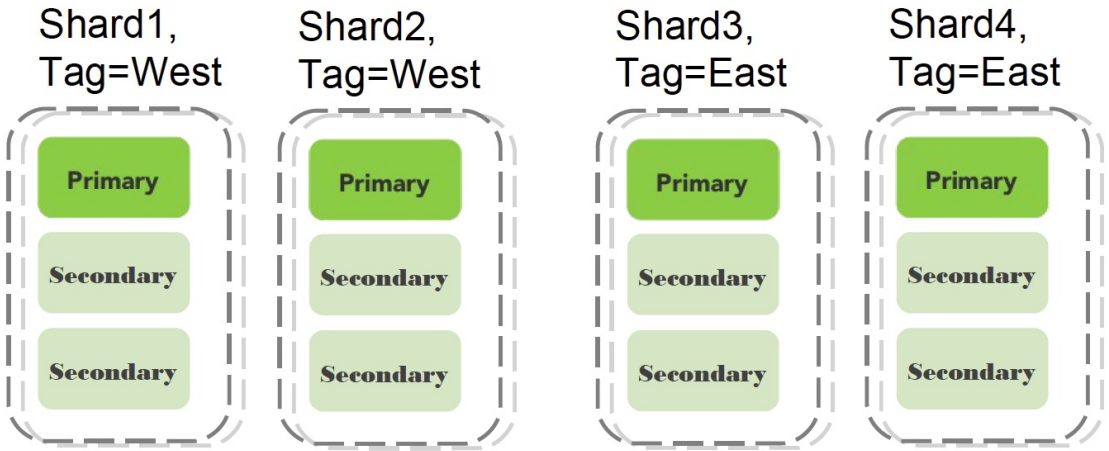
Tag Sharding Example

Collection: Users, **Shard Key:** {uId, regionCode}

Tag	Start	End
West	MinKey, MinKey	MaxKey,50
East	MinKey, 50	MaxKey, MaxKey

Assign Regions 1-50 to the West

Assign Regions 51-100 to the East



Which Sharding to use?

Usage	Required Strategy
Scale	Range or Hash
Geo-Locality	Tag-aware
Hardware Optimization	Tag-aware
Lower Recovery Times	Range or Hash

CAP Theorem and Mongo

Focus on Consistency and Partition tolerance

- **C**onsistency
  - all replicas contain the same version of the data
- **A**vailability
  - system remains operational on failing nodes
- **P**artition tolarence
  - multiple entry points
  - system remains operational on system split