Implementing Time Series

Tony Li
Consulting Manager
MongoDB

SEPT 7

Working With Time Series Collections

Terminology & Concepts

Launched with 5.0

Creating a Time Series Collection

```
db.createCollection("weather", {
    timeseries: {
        timeField: "timestamp",
        metaField: "sensorId",
        granularity: “minutes”
    },
    expireAfterSeconds: 9000

})
```

The timeField is the only required parameter for a Time Series collection

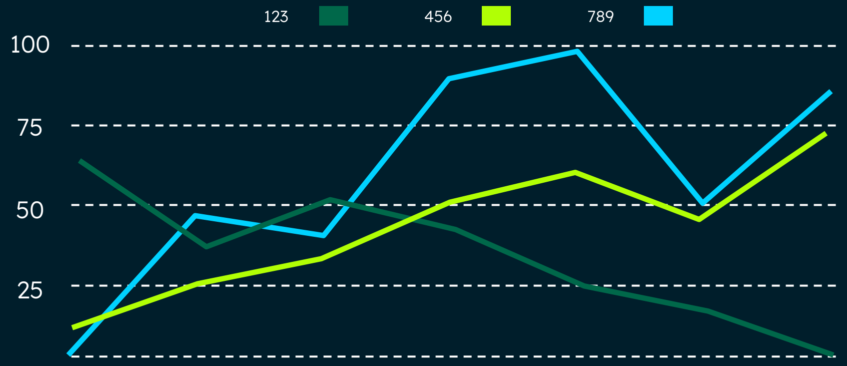
TO CREATE A TIME SERIES COLLECTION, USE THE **timeseries** OPTION

Terminology & concepts: metaField

```
> db.createCollection ("weather", { timeseries: { ..., metaField: “sensorId” } } )
```

```
{
  "sensorId": 123,
  “timestamp”: ISODate(“...”),
  “temperature”: 47.0
},
{
  "sensorId": 456,
  “timestamp”: ISODate(“...”),
  “temperature”: 69.8
},
{
  "sensorId": 789,
  “timestamp”: ISODate(“...”),
  “temperature”: 97.0
}
```

- Label or tag that uniquely identifies a time series
- Never/rarely changes over time

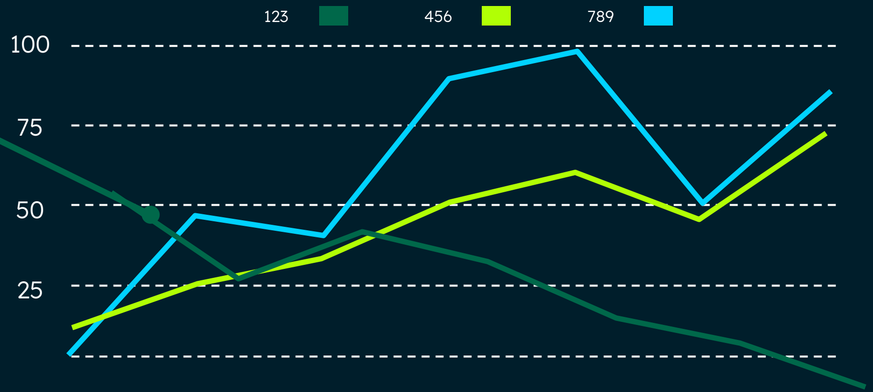


Terminology & concepts: metric

```
> db.createCollection ("weather", { timeseries: { ..., metaField: “sensorId” } } )
```

```
{
  "sensorId": 123,
  “timestamp”: ISODate(“..."),
  “temperature”: 47.0
},
{
  "sensorId": 456,
  “timestamp”: ISODate(“..."),
  “temperature”: 69.8
},
{
  "sensorId": 789,
  “timestamp”: ISODate(“..."),
  “temperature”: 97.0
}
```

- A set of related key-value pairs at a specific time
- Any other fields except metadata and time



Terminology & concepts: measurement

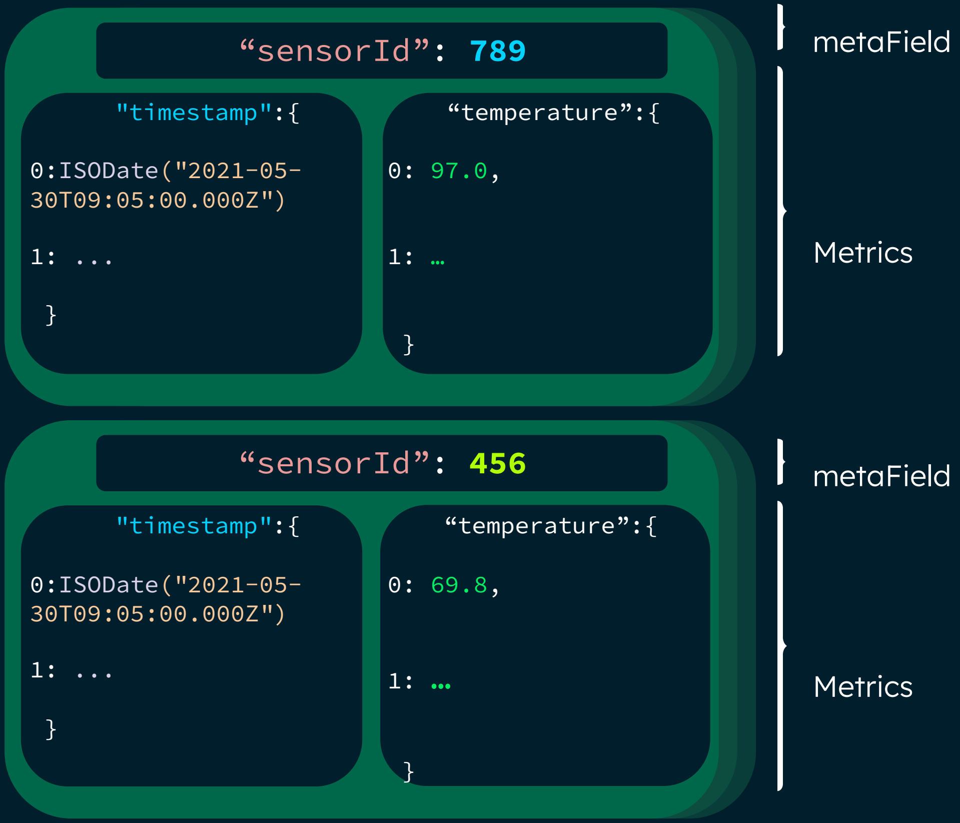
```
> db.createCollection ("weather", { timeseries: { ..., metaField: “sensorId” } } )
```

```
{
  "sensorId": 123,
  “timestamp”: ISODate(“..."),
  “temperature”: 47.0
},
{
  "sensorId": 456,
  “timestamp”: ISODate(“..."),
  “temperature”: 69.8
},
{
  "sensorId": 789,
  “timestamp”: ISODate(“..."),
  “temperature”: 97.0
}
```

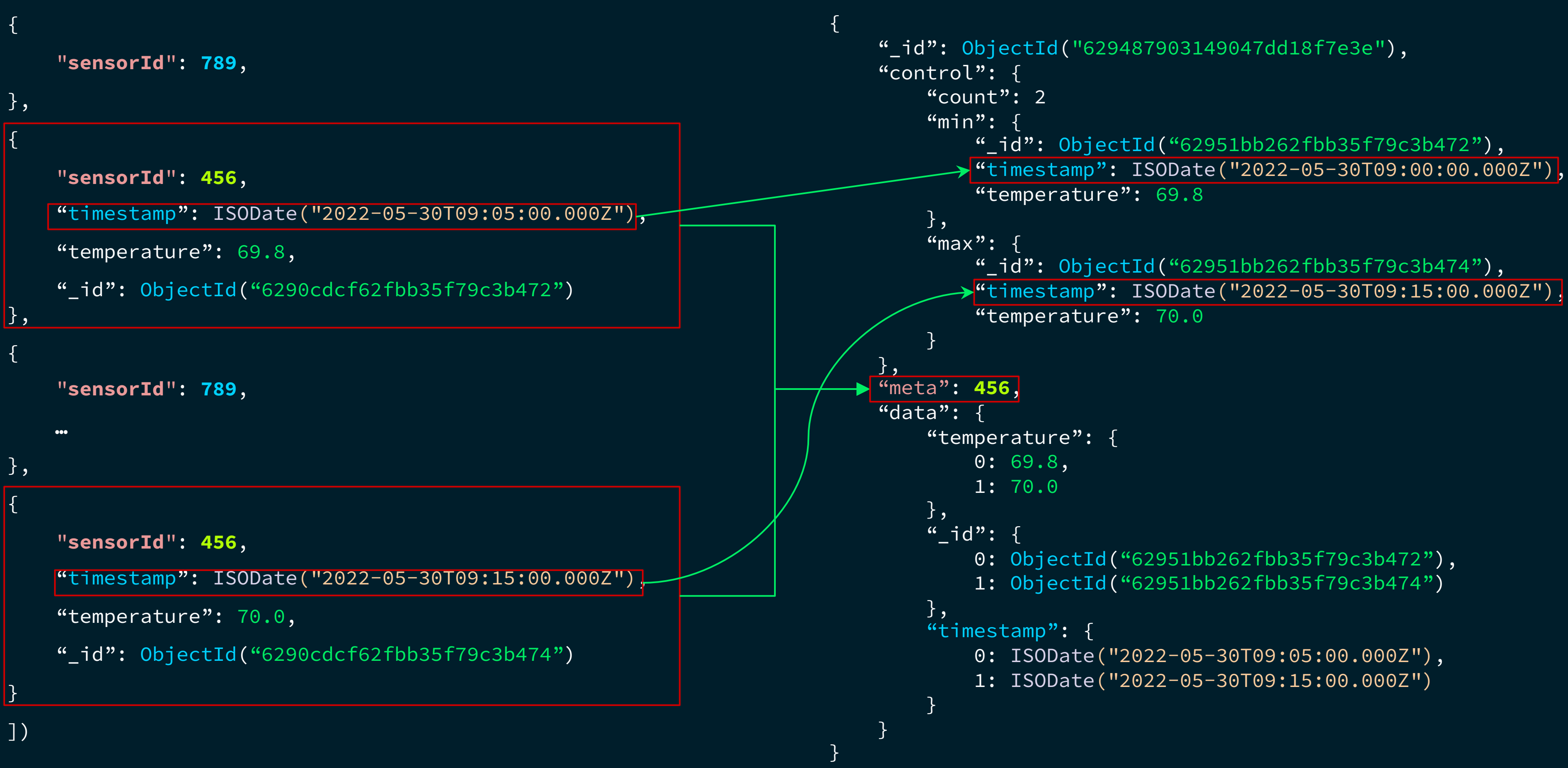
- A user-facing document inserted in a time-series collection

Terminology & concepts: bucket

```
> db.weather.insertMany([

{
    "sensorId": 789,
    "timestamp": ISODate("2022-05-30T09:05:00.000Z"),
    “temperature”: 97.0
}, {
    "sensorId": 456,
    "timestamp": ISODate("2022-05-30T09:05:00.000Z"),
    “temperature”: 69.8

} •••
)]
```



```
“sensorId”: 789                                   metaField

"timestamp":{                 “temperature”:{
0:ISODate("2021-05-           0: 97.0,
30T09:05:00.000Z")                                Metrics
1: ...                        1: …
}                             }

“sensorId”: 456                                   metaField

"timestamp":{                 “temperature”:{
0:ISODate("2021-05-           0: 69.8,
30T09:05:00.000Z")                                Metrics
1: ...                        1: …
}                             }
```



```
> db.weather.insertMany([

{
    "sensorId": 789,
},
{
    "sensorId": 456,
    “timestamp”: ISODate("2022-05-30T09:05:00.000Z"),
    “temperature”: 69.8,
    “_id”: ObjectId(“6290cdcf62fbb35f79c3b472”)
},
{
    "sensorId": 789,
    …
},
{
    "sensorId": 456,
    “timestamp”: ISODate("2022-05-30T09:15:00.000Z"),
    “temperature”: 70.0,
    “_id”: ObjectId(“6290cdcf62fbb35f79c3b474”)
}
])
```

```
{
    “_id”: ObjectId("629487903149047dd18f7e3e"),
    “control”: {
        “count”: 2
        “min”: {
            “_id”: ObjectId(“62951bb262fbb35f79c3b472”),
            “timestamp”: ISODate("2022-05-30T09:00:00.000Z"),
            “temperature”: 69.8
        },
        “max”: {
            “_id”: ObjectId(“62951bb262fbb35f79c3b474”),
            “timestamp”: ISODate("2022-05-30T09:15:00.000Z"),
            “temperature”: 70.0
        }
    },
    “meta”: 456,
    “data”: {
        “temperature”: {
            0: 69.8,
            1: 70.0
        },
        “_id”: {
            0: ObjectId(“62951bb262fbb35f79c3b472”),
            1: ObjectId(“62951bb262fbb35f79c3b474”)
        },
        “timestamp”: {
            0: ISODate("2022-05-30T09:05:00.000Z"),
            1: ISODate("2022-05-30T09:15:00.000Z")
        }
    }
}
```

Terminology & Concepts: granularity

Market Data

“seconds”

1 minute → 1 hour

Fleet monitoring

“minutes”

1 hour → 1 day

Weather sensors

“hours”

1 day → 30 days

Granularity controls the time span in which measurements with the **same** metaField values can be stored and colocated as **one** bucket on disk

Launched with 7.0

New Granularity Option

Fixed Time Interval Bucketing using a new granularity alternate option: `bucketMaxSpanSeconds`

Use Cases

- Collecting fixed/regular time-series data
- Query intervals are also fixed
- Specify a more defined granularity

Terminology & concepts: expireAfterSeconds

Replaces TTL indexes

Optimized delete performance

Can be changed using `collMod`

Data Modeling Exercise

Scenario:

You collect streaming market data trade streams for indexes, e.g. NASDAQ

You want to aggregate trade information such as price/quantity and events by the symbol, e.g. “MDB”

Data Modeling Exercise

Field Name	Description
eventType	Event type, e.g. “trade”
eventTime	Time of the event
symbol	Security symbol, e.g. “MDB”
tradeId	Unique identifier for a trade
price	Price of the underlying security traded
quantity	Quantity of the security traded
buyerOrderId	Unique order identifier for buyer
sellerOrderId	Unique order identifier for seller

Good metaField

```
meta: {
            symbol: “MDB”
}
```

Bad metaField

```
meta: {
            symbol: “MDB”,
      tradeId: 1235657471234
}
```

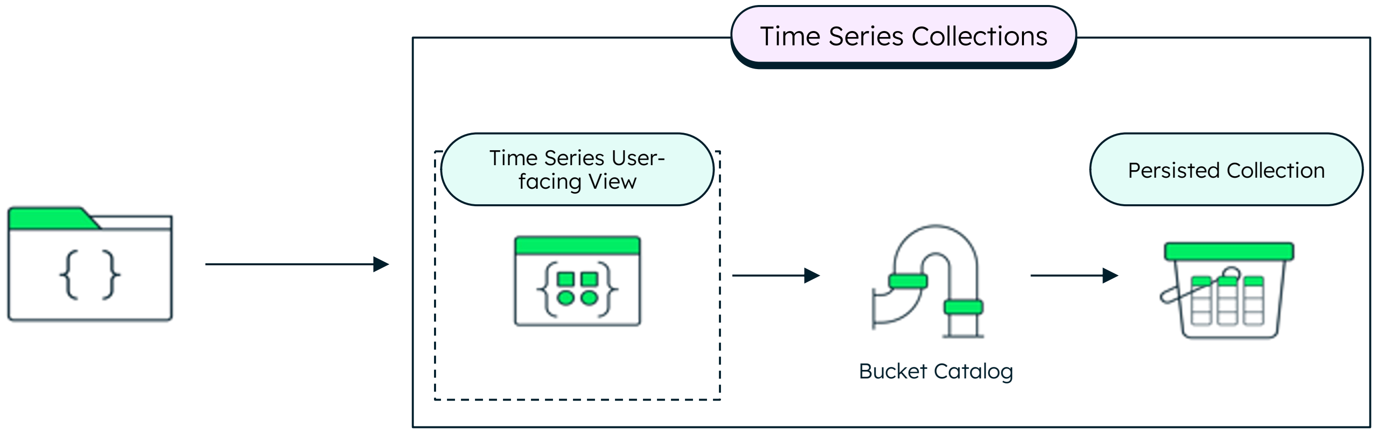
Terminology & concepts: cardinality

```
» db.createCollection ("weather", { timeseries: { ..., metaField: "meta" } } )

{
   "meta": {sensorId: 123, locale: "en-us"},
   "timestamp": ISODate("..."),
   "temperature": 47.0
}
```

- Unique number of "things" aka "time-series"
- The total number of unique combination of values for the metaField

How Time Series Collections Work

How do Time Series Collections Work?



Create a Time Series Collection

Write data as single documents

Data is automatically persisted into an optimized columnar storage format

Launched with
5.0

Inserting data into a Time Series Collection

```
> db.weather.insertMany([
{
    "sensorId": 789,
    "timestamp": ISODate("2022-05-30T09:05:00.000Z"),
    “temperature”: 97.0
}, {
    "sensorId": 456,
    “timestamp”: ISODate("2022-05-30T09:05:00.000Z"),
    “temperature”: 69.8
}, {
    "sensorId": 789,
    "timestamp": ISODate("2022-05-30T09:15:00.000Z"),
    “temperature”: 97.0
}, {
    "sensorId": 456,
    "timestamp": ISODate("2022-05-30T09:15:00.000Z"),
    “temperature”: 70.0
}
)]
```

Launched with 5.0

Querying a Time Series Collection

```
> db.weather.find()
```

```
{
    "sensorId": 789,
    "timestamp": ISODate("2022-05-30T09:05:00.000Z"),
    “temperature”: 97.0
}, {
    "sensorId": 456,
    "timestamp": ISODate("2022-05-30T09:05:00.000Z"),
    “temperature”: 69.8
}, {
    "sensorId": 789,
    "timestamp": ISODate("2022-05-30T09:15:00.000Z"),
    “temperature”: 97.0
}, {
    "sensorId": 456,
    "timestamp": ISODate("2022-05-30T09:15:00.000Z"),
    “temperature”: 70.0
}
```

Launched with 5.0

Time Series Collection Bucketing Catalog

A bucket is a group of documents stored together as one document with the same metaField for a time span

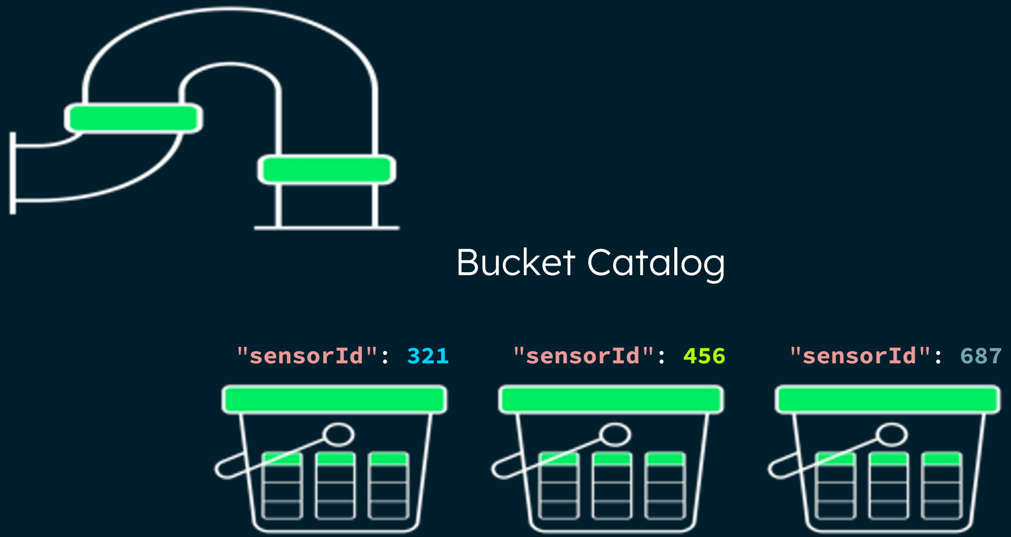
Transforms an insert of a measurement into a Time Series Collection into an insert or update on the bucket collection

Synchronizes and batches concurrent updates to the same bucket

Compresses data in-memory

Allocates 2.5% of total physical RAM for “open” buckets

```
> db.weather.insertMany([

{
    "sensorId": 321,
    “timestamp”: ISODate("2022-05-30T09:00:00.000Z"),
    “temperature”: 69.6
},
{
    "sensorId": 456,
    “timestamp”: ISODate("2022-05-30T09:00:00.000Z"),
    “temperature”: 85.8
},
{
    "sensorId": 687,
    “timestamp”: ISODate("2022-05-30T09:00:00.000Z"),
    “temperature”: 70.0
} •••
])
```



Improved Support for MongoDB Time Series

Since MongoDB 5.0, there have been a significant number of features and capabilities added to improve Time Series collections

Time Series 7.0 updates

Enhanced Scalability

Update/Delete Support

Enhanced Performance

Partial TTL Indexes

Enhanced and Less Expensive Scalability (Launched with 7.0)

Time series collections can now handle more distinct time series with less resources, even up to millions of series on modest machines

Launched with 7.0

Enhanced Scalability For High Cardinality Workloads

Scaling for high-cardinality time-series workloads is now easier, less expensive and more performant

Avoiding premature bucket closure with archival-based and query-based reopening approaches for high-cardinality workloads

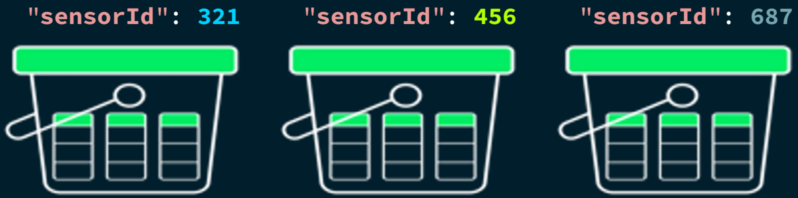
Reducing the impact of drop commands and chunk migrations

More flexible bucketing

Improved performance and bucketing

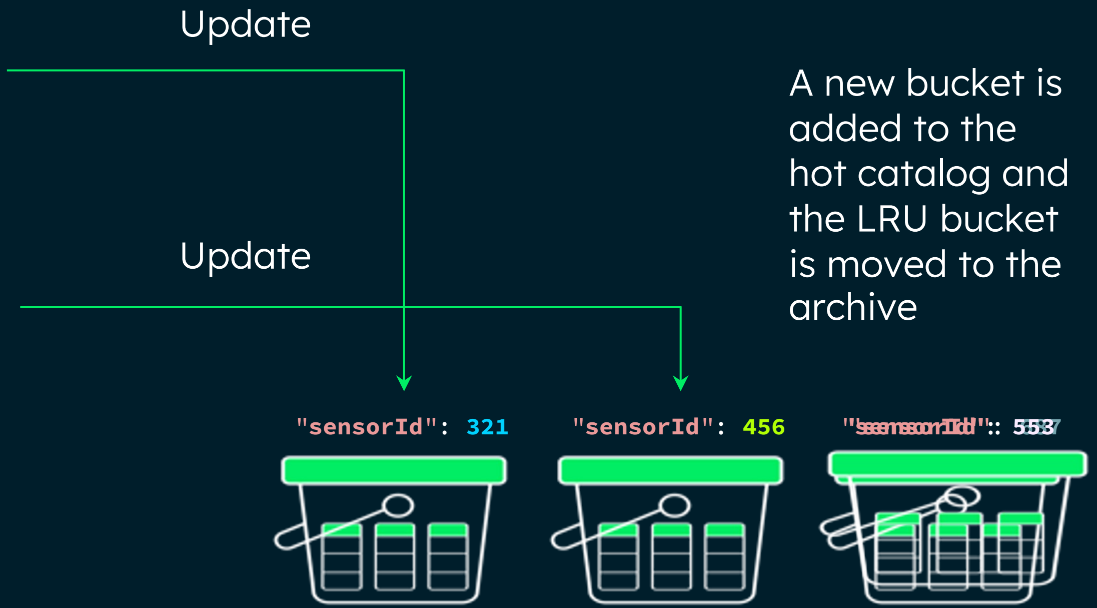
```
> db.weather.insertMany([

{
    "sensorId": 321,
    “timestamp”: ISODate("2022-05-30T09:00:00.000Z"),
    “temperature”: 69.6
},
{
    "sensorId": 456,
    “timestamp”: ISODate("2022-05-30T09:00:00.000Z"),
    “temperature”: 85.8
},
{
    "sensorId": 687,
    “timestamp”: ISODate("2022-05-30T09:00:00.000Z"),
    “temperature”: 70.0
} •••
])
```

Bucket Catalog



Five Minutes Later...

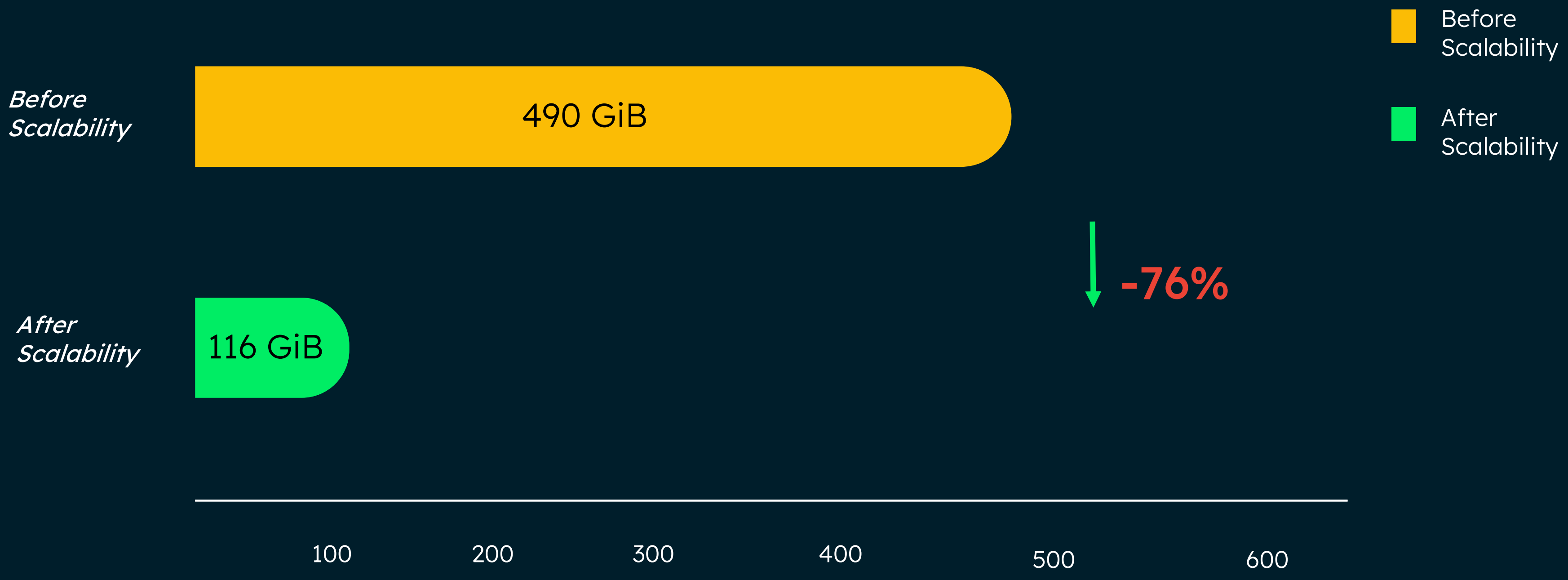
```
> db.weather.insertMany([


{
    "sensorId": 321,
    “timestamp”: ISODate("2022-05-30T09:05:00.000Z"),
    “temperature”: 69.2
},
{
    "sensorId": 456,
    “timestamp”: ISODate("2022-05-30T09:05:00.000Z"),
    “temperature”: 86.8
},
{
    "sensorId": 553,
    “timestamp”: ISODate("2022-05-30T09:05:00.000Z"),
    “temperature”: 70.0
}
])
```



A new bucket is added to the hot catalog and the LRU bucket is moved to the archive

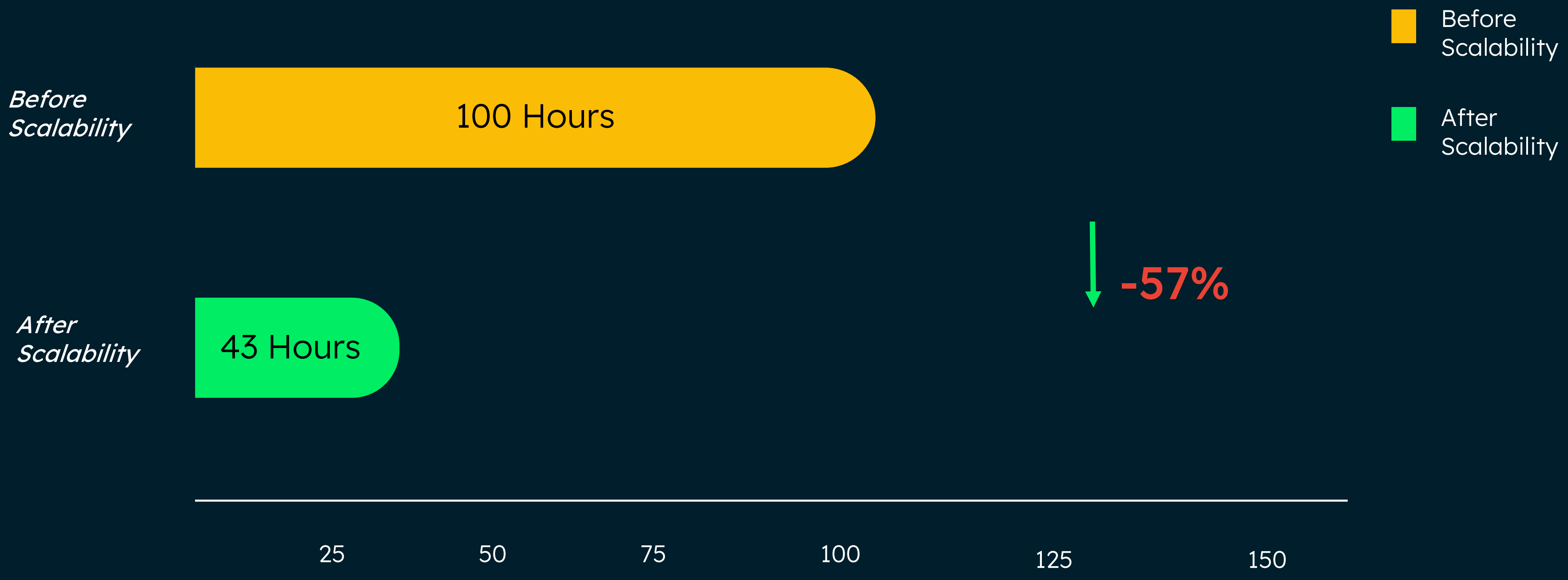
Enhanced Scalability

Data Size (GiB)



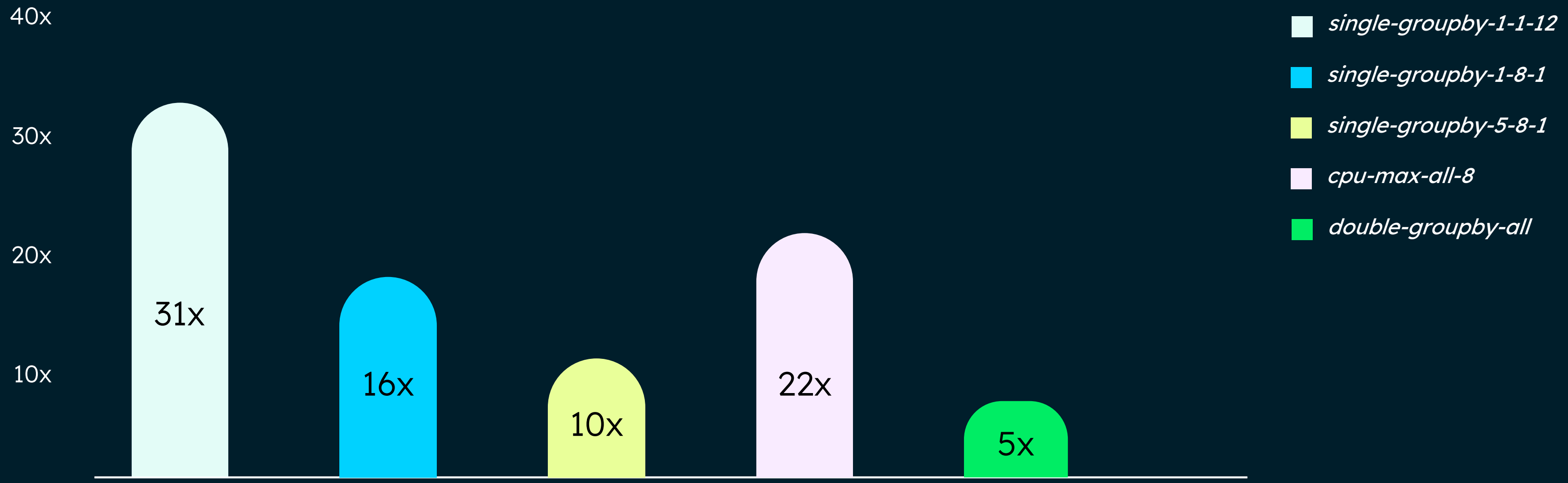
Enhanced Scalability

Load Performance (Hours)

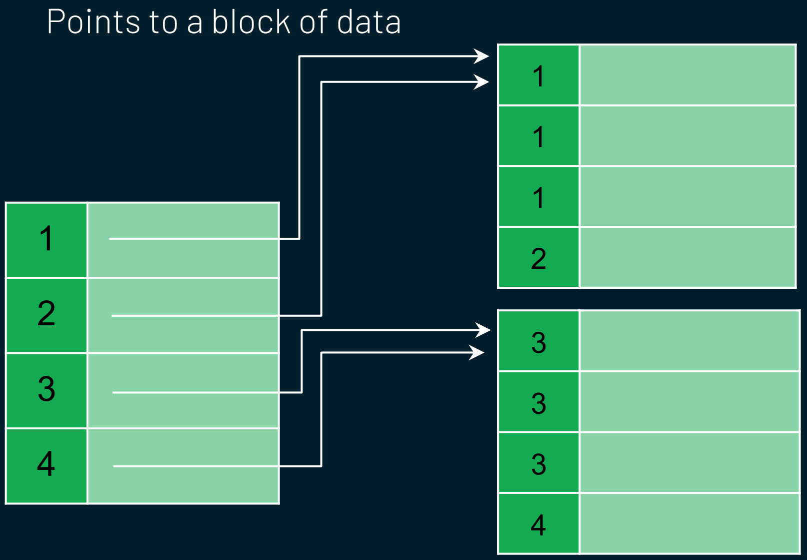


Enhanced Scalability

Query Performance (Avg. Latency)



Time Series Collection Indexes



Reduces Index Sizes

- “Buckets”, i.e. groups of documents, are indexed, not individual documents
- One unique identifier per bucket results in an overall reduction in index size of hundreds of times

Clustered Index on Time

- System generated clustered index on the “bucket” time which orders data on disk by time
- Adjacent buckets can be stored in the same page
- Reduces the cost of scans on time

New Time Series Features (7.0)

Time Series Collections

Support for arbitrary deletes and updates* across any fields including updates*/deletes of single or multiple records and findAndModify*

Other Enhancements

- Enhanced Scalability
- Performance Optimizations
- Partial TTL Indexes

**Support for Updates coming soon*

Improved Support for MongoDB Time Series

Since MongoDB 5.0, there have been a significant number of features and capabilities added to improve Time Series collections

Time Series 7.0 updates

Enhanced Scalability

Update/Delete Support

Enhanced Performance

Partial TTL Indexes

Full Update/Delete Support (Launched with 7.0)

Support for arbitrary updates and deletes across all fields including singleton updates, multi-deletes and find & modify

update(), delete(), updateMany(), deleteMany(), findAndModify()

**Support for updates coming soon*

Improved Support for MongoDB Time Series

Since MongoDB 5.0, there have been a significant number of features and capabilities added to improve Time Series collections

Time Series 7.0 updates

Enhanced Scalability

Update/Delete Support

Enhanced Performance

Partial TTL Indexes

Enhanced Query Performance (Launched with 7.0)

Continued emphasis on specialized optimizations taking advantage of the columnar format and bucketing of data.

Time-based grouping optimizations including Streaming Group operations, reducing the time to first batch, and avoiding blocking operations and hash table operations

Improved Support for MongoDB Time Series

Since MongoDB 5.0, there have been a significant number of features and capabilities added to improve Time Series collections

Time Series 7.0 updates

Enhanced Scalability

Update/Delete Support

Enhanced Performance

Partial TTL Indexes

Partial TTL Indexes (Launched with 7.0)

Support for partial TTL indexes to expire data on additional criteria aside from time in the metaField