Ledger as an auditing solution for cloud database

By CloudChain

Team Members

Akash Malla

Sharon Subathran

Neha Soma

Sneha Mule

Outline

- Introduction
- Implementation
- Demo
- Conclusion
- Challenges
- Take Away

Introduction

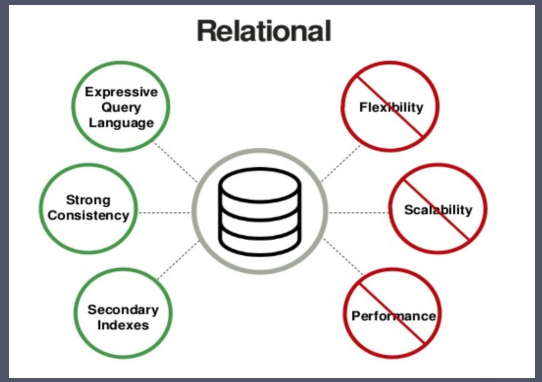
Aim:

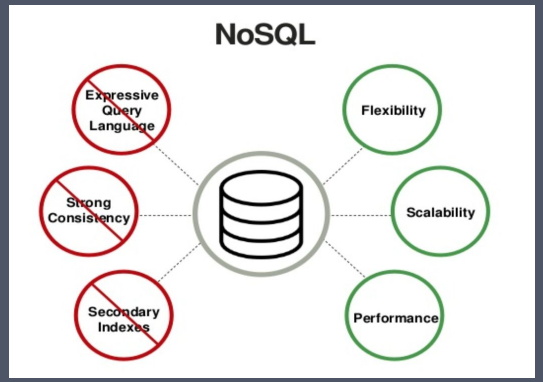
- To improve the security and consistency of data in a NoSQL database namely mongoDB.
- With further research, we felt that we could not contribute anymore than the existing solution (Crypt-NoSQL,).
- To provide a concise and clean platform to audit logs in a NoSQL database such as mongoDB.

Problem Statement

- Typically, many clients have to extract data from database logs to analyze performance issues.
- Reviewing and formatting log data is time consuming, if not, more time consuming than analyzing formatted data using a visualization utility or machine learning algorithm
- We wanted to eliminate the heavy lifting of formatting and capture all active operation related data in a single collection for any client to be able to query.

Why MongoDB?





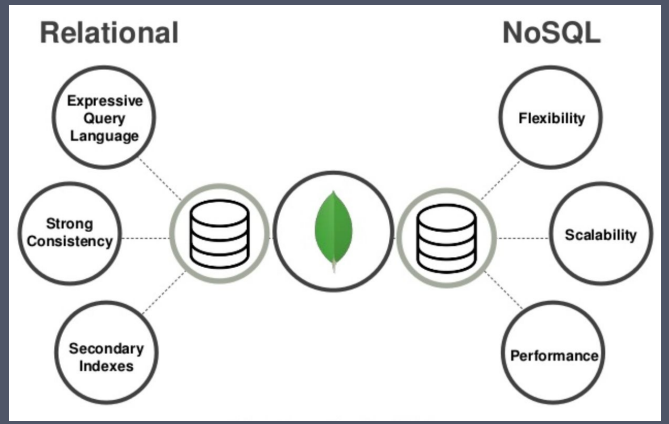


Image Ref : *when-to-use-mongodb-when-you-should-not*

Current MongoDB Tools

- MongoDB Atlas:
  - It takes data in JSON format.
  - It creates a cluster and gives visual representation.

- Coralogix:
  - It has a button for loggregator.
  - It gives pattern which collects same log clusters.

- AWS cloudWatch:
  - It can set alert based on threshold set.
  - It can store log data from multiple resource LogGroup

Distributed Ledger

- Distributed Ledger technology (DLT) may have started off as the basis for bitcoin, but it already promises to be much more than a cryptocurrency.
- Records/logs are stored one after the other in a continuous ledger, rather than sorted into blocks, but they can only be added when the participants reach a quorum.
- Currently, there are a number of digital ledgers that are emerging. Among them are ledgers which don't communicate with each other such as **Hyperledger Fabric, R3's Corda, Ripple, and more.**

Distributed Ledger Classification

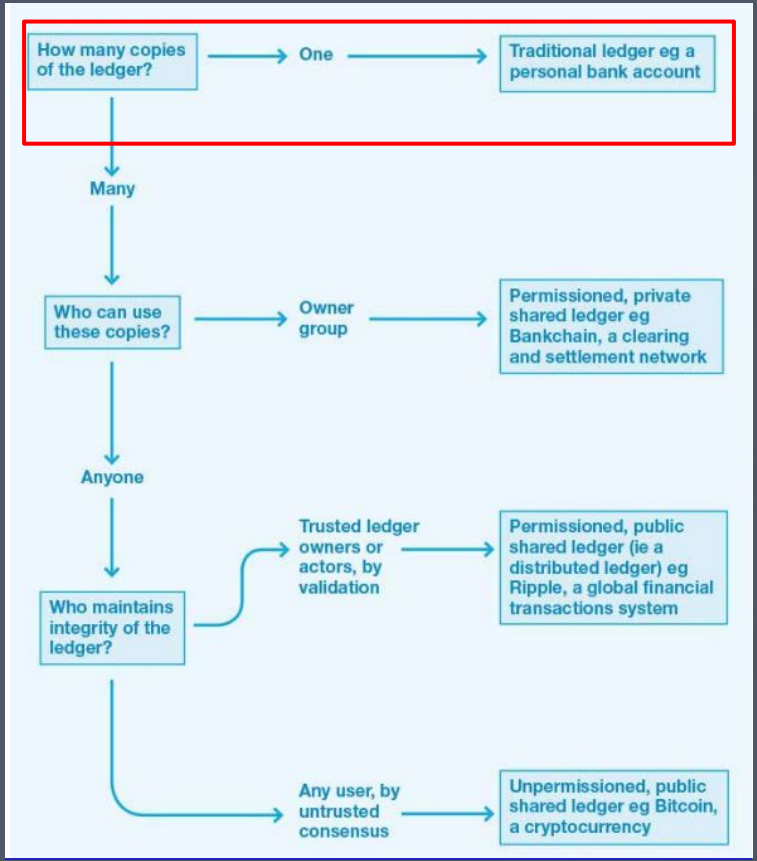


Image Ref : *Distributed Ledger Technology: beyond block-chain*

Implementation Overview

- In our architecture, we built the ledger as a way to audit the mongodb logs instead of having large text files.

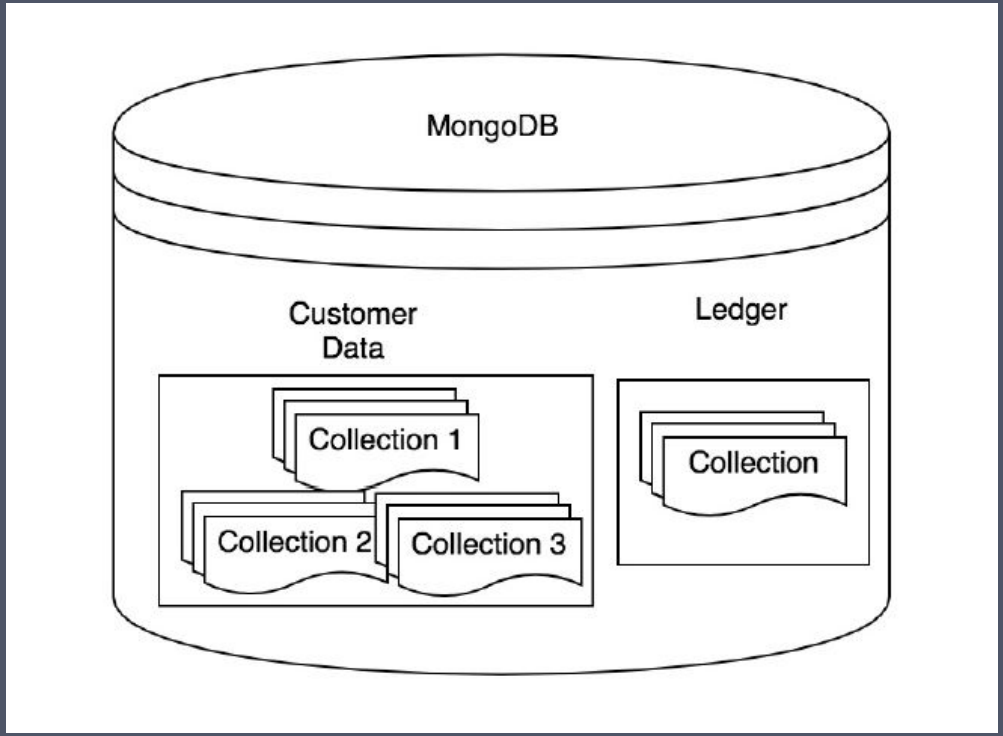
- This ledger is a collection, so it consists of objects which are easier to analyze and find valuable data for a consumer in order to determine what kind of queries are taking time.

Implementation Overview (contd..)

- The ledger facilitates a way to ensure that the right data is fetched and added into the ledger by using multiple mongoDB commands.

- List of operations in the ledger can be accessed easily from the mongodb querying platform by all clients.

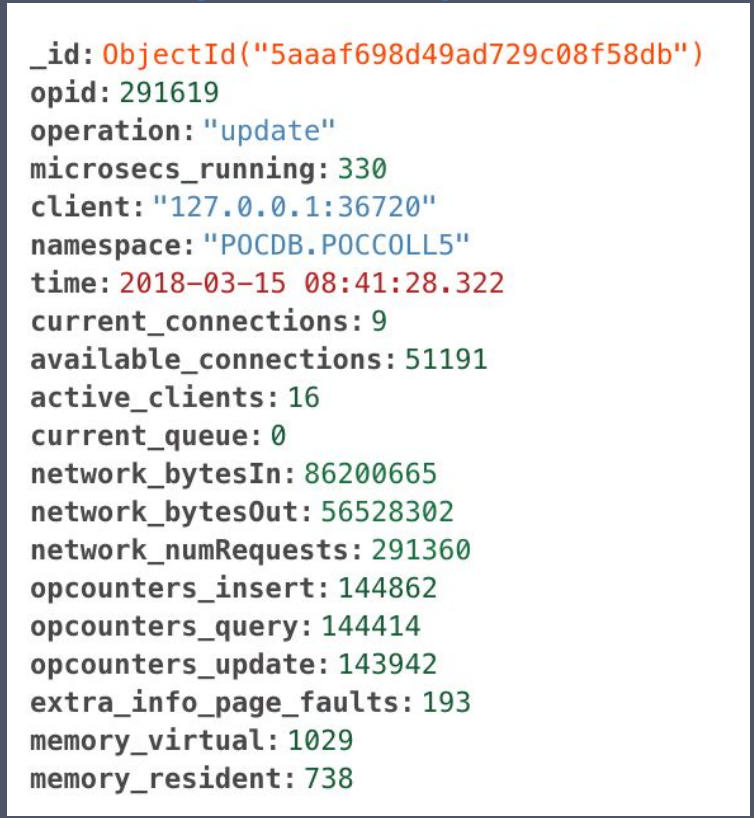
Our Design


MongoDB
Customer Data
Collection 1
Collection 2
Collection 3
Ledger
Collection

Ledger Parameters used:

- Operation id and type
- Microseconds Running
- Client IP address
- Namespace/Collection on which operation is running on
- Current and available connections
- Network byte input and byte output
- Network number of requests
- Total number of inserts, queries, updates at the moment
- Total number of updates at the moment
- Number of page faults
- Virtual and resident memory of a process

Image of our ledger:



Tools we used

- POCDriver.jar (https://github.com/johnlpage/POCDriver) - For load testing mongoDB instance.
- Amazon AWS EC2 (Ubuntu 16.04.3)
- Mongo Compass - For visualizing ledger data and mongodb data other than mongo shell
- Pymongo API in order to connect to mongodb in python
- Mongo shell

Implementation in Detail

Below is how we kick start the load test:

```
ubuntu@ip-172-31-27-35:~$ java -jar POCDriver.jar -k 20 -i 20 -u 20 -d 600 -t 3 -e -y 10
MongoDB Proof Of Concept - Load Generator
Worker thread 0 Started.
Worker thread 1 Started.
Worker thread 2 Started.
------------------------
After 10 seconds, 15165 new records inserted - collection has 15165 in total
1387 inserts per second since last report 0.00 % in under 50 milliseconds
144 keyqueries per second since last report 100.00 % in under 50 milliseconds
1453 updates per second since last report 90.48 % in under 50 milliseconds
0 rangequeries per second since last report 100.00 % in under 50 milliseconds

------------------------
After 20 seconds, 36645 new records inserted - collection has 36645 in total
2148 inserts per second since last report 0.00 % in under 50 milliseconds
210 keyqueries per second since last report 100.00 % in under 50 milliseconds
2157 updates per second since last report 90.36 % in under 50 milliseconds
0 rangequeries per second since last report 100.00 % in under 50 milliseconds
```

Mongostat to check performance

```
ubuntu@ip-172-31-27-35:~$ mongostat
insert query update delete getmore command % dirty % used flushes vsize   res qr|qw ar|aw netIn netOut conn                 time
    *0    *0     *0     *0       0     14|0     0.0    0.1       0  280M 53.0M   0|0   0|0 1.99k  52.7k    4 2018-03-15T21:09:53Z
    *0    *0     *0     *0       0     18|0     0.0    0.1       0  280M 53.0M   0|0   0|0 1.67k   286k    3 2018-03-15T21:09:54Z
    *0    *0     *0     *0       0     19|0     0.0    0.1       0  280M 53.0M   0|0   0|0 1.74k   267k    2 2018-03-15T21:09:55Z
    *0    *0     *0     *0       0     14|0     0.0    0.1       0  280M 53.0M   0|0   0|0 1.99k  52.1k    4 2018-03-15T21:09:56Z
   339   416    331     *0       0     34|0     0.1    1.9       0  304M 73.0M   0|0   1|0  216k   343k    5 2018-03-15T21:09:57Z
  1016  1100   1181     *0       0     26|0     0.5    5.2       0  339M  106M   0|0   1|0  636k   551k    7 2018-03-15T21:09:58Z
  1429  1258   4193     *0       0     10|0     1.1    7.7       0  364M  130M   0|0   0|0 1.08m   389k    9 2018-03-15T21:09:59Z
   864  1046  13980     *0       0     24|0     1.3    8.6       0  374M  140M   0|0   0|1 1.67m   592k    5 2018-03-15T21:10:00Z
  1854  1709   1949     *0       0     10|0     2.0   10.3       0  391M  157M   0|0   0|1 1.10m   452k    7 2018-03-15T21:10:01Z
  1621  1652   1667     *0       0     47|0     2.5   11.5       0  405M  170M   0|0   0|0  977k  1.11m    5 2018-03-15T21:10:02Z
insert query update delete getmore command % dirty % used flushes vsize   res qr|qw ar|aw netIn netOut conn                 time
  1896  1828   1793     *0       0     28|0     3.1   12.7       0  418M  182M   0|0   0|0 1.11m   692k    9 2018-03-15T21:10:03Z
  1792  1822   1636     *0       0     24|0     3.7   13.9       0  430M  194M   0|0   0|0 1.05m   817k    7 2018-03-15T21:10:04Z
  2032  1783   1997     *0       0     20|0     4.3   15.0       0  443M  206M   0|0   0|0 1.17m   585k    7 2018-03-15T21:10:05Z
  1727  2079   1809     *0       0     18|0     5.0   16.2       0  456M  218M   0|0   0|0 1.08m   744k    8 2018-03-15T21:10:06Z
  2029  1815   2035     *0       0     10|0     5.2   17.4       0  468M  229M   0|0   0|0 1.18m   459k    5 2018-03-15T21:10:07Z
  2080  2143   2043     *0       0     27|0     5.3   18.6       0  482M  242M   0|0   0|0 1.24m   822k    5 2018-03-15T21:10:08Z
  2220  2159   2177     *0       0     28|0     5.3   19.9       0  495M  256M   0|0   0|0 1.31m   861k    5 2018-03-15T21:10:09Z
  2185  2003   2160     *0       0     10|0     4.9   21.0       0  508M  268M   0|0   0|0 1.28m   503k    8 2018-03-15T21:10:10Z
  2090  2282   2148     *0       0     24|0     5.1   22.3       0  522M  281M   0|0   1|0 1.28m   877k    5 2018-03-15T21:10:11Z
  2063  1924   2063     *0       0      6|0     4.8   23.4       0  535M  293M   0|0   1|0 1.21m   480k    6 2018-03-15T21:10:12Z
insert query update delete getmore command % dirty % used flushes vsize   res qr|qw ar|aw netIn netOut conn                 time
  2050  2102   2067     *0       0     30|0     4.8   24.6       0  548M  304M   0|0   0|0 1.23m   826k    5 2018-03-15T21:10:13Z
  2049  2059   2007     *0       0     16|0     4.7   25.7       0  561M  316M   0|0   0|0 1.22m   548k    8 2018-03-15T21:10:14Z
  2142  2141   2173     *0       0     21|0     4.7   26.8       0  573M  327M   0|0   0|0 1.28m   706k    7 2018-03-15T21:10:15Z
  2425  2218   2210     *0       0     18|0     5.2   28.2       0  588M  341M   0|0   0|0 1.39m   658k    5 2018-03-15T21:10:16Z
  1994  2165   2052     *0       0     32|0     5.2   29.2       0  600M  352M   0|0   1|0 1.22m   832k    5 2018-03-15T21:10:17Z
  2723  2444   2723     *0       0     22|0     6.2   30.6       0  614M  364M   0|0   0|0 1.59m   846k    9 2018-03-15T21:10:18Z
  2403  2460   2451     *0       0     15|0     5.7   31.9       0  627M  376M   0|0   0|0 1.44m   656k    5 2018-03-15T21:10:19Z
  2651  2668   2548     *0       0     17|0     5.6   33.2       0  642M  389M   0|0   0|0 1.57m   661k   11 2018-03-15T21:10:20Z
  2629  2664   2664     *0       0     23|0     5.5   34.6       0  657M  402M   0|0   0|0 1.57m  1.03m    5 2018-03-15T21:10:21Z
  2690  2701   2720     *0       0     31|0     5.3   35.9       0  672M  416M   0|0   1|1 1.60m   936k    7 2018-03-15T21:10:22Z
```

Code

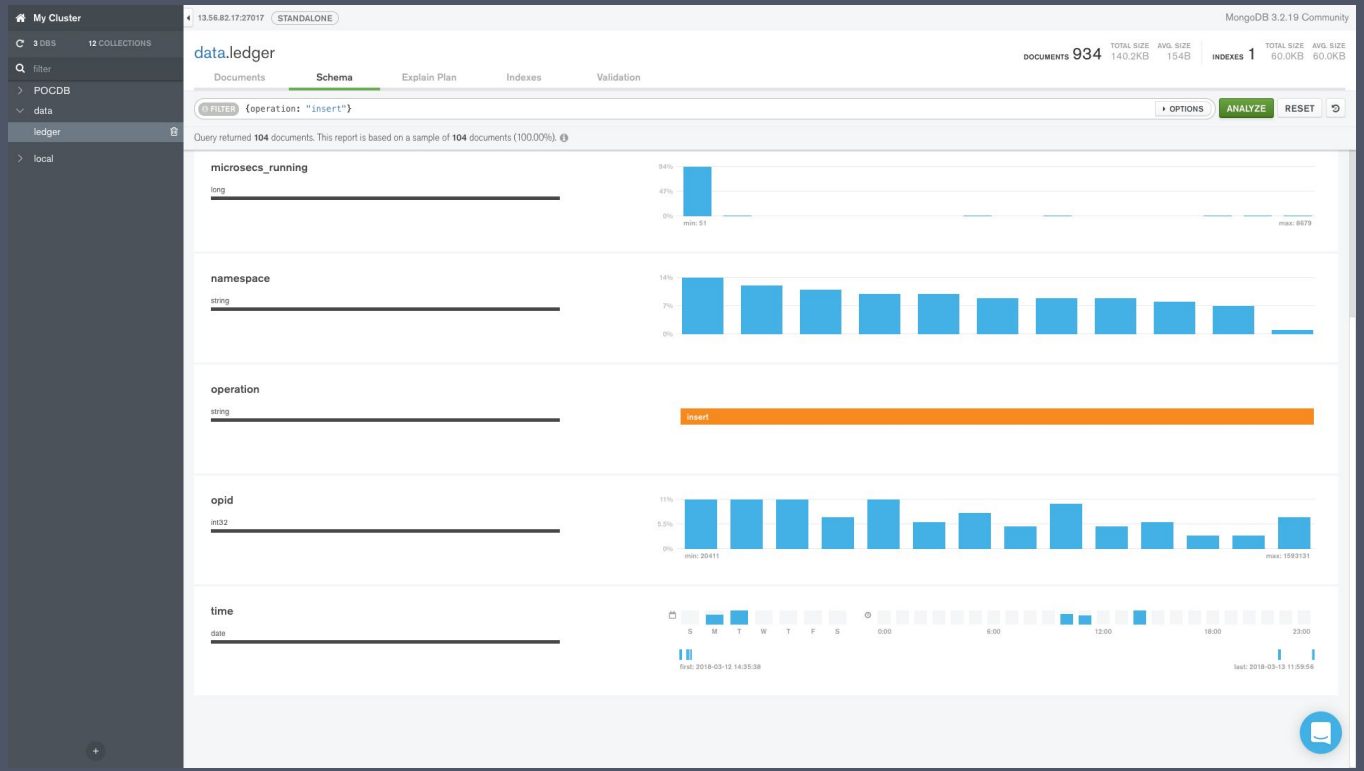
- mongodb_conn() function tries to connect to mongodb server in 10 seconds.

```
import pymongo
import schedule
import time
import datetime
import traceback

def mongodb_conn():
    try:
        maxSevSelDelay=10000
        client = pymongo.MongoClient("13.56.82.17",serverSelectionTimeoutMS=maxSevSelDelay)
        client.server_info()
    except pymongo.errors.ConnectionFailure as err1:
        print("Could not connect to server, connection failure:",err1)
    except pymongo.errors.ServerSelectionTimeoutError as err2:
        print("Could not connect to server, ServerSelectionTimeoutError:",err2)
    return client
```

```
def job():
    conn = mongodb_conn()
    #print("connected successfully")
    if conn is None:
        # no connection, exit early
        print("not connected to mongodb!")
        return

    #print("I'm working...")
    try:
        db = conn.data
        input=db.current_op(True)

        for j in input['inprog']:

            if j['active']==True:
                if 'microsecs_running' in j and j['microsecs_running']>50 and j['microsecs_running']<15000:
                        #print(j)
                        newJ=dict()
                        if 'opid' in j:
                            newJ['opid']=j['opid']
                        if 'op' in j:
                            newJ['operation']=j['op']
                        if 'microsecs_running' in j:
                            newJ['microsecs_running']=j['microsecs_running']
                        if 'client' in j:
                            newJ['client']=j['client']
                        if 'ns' in j:
                            newJ['namespace']=j['ns']
                        newJ['time']=datetime.datetime.now()
                        server_status=db.command({"serverStatus":1})
                        newJ['current_connections']=server_status['connections']['current']
                        newJ['available_connections']=server_status['connections']['available']
                        newJ['active_clients']=server_status['globalLock']['activeClients']['total']
                        newJ['current_queue']=server_status['globalLock']['currentQueue']['total']
                        newJ['network_bytesIn']=server_status['network']['bytesIn']
                        newJ['network_bytesOut']=server_status['network']['bytesOut']
                        newJ['network_numRequests']=server_status['network']['numRequests']
                        newJ['opcounters_insert']=server_status['opcounters']['insert']
                        newJ['opcounters_query']=server_status['opcounters']['query']
                        newJ['opcounters_update']=server_status['opcounters']['update']
```

```
                    newJ['network_numRequests']=server_status['network']['numRequests']
                    newJ['opcounters_insert']=server_status['opcounters']['insert']
                    newJ['opcounters_query']=server_status['opcounters']['query']
                    newJ['opcounters_update']=server_status['opcounters']['update']
                    newJ['extra_info_page_faults']=server_status['extra_info']['page_faults']
                    newJ['memory_virtual']=server_status['mem']['virtual']
                    newJ['memory_resident']=server_status['mem']['resident']
                    print(newJ)
                    #print("inside\n")

                    db.ledgerData.insert_one(newJ)

    except:
        traceback.print_exc()
        exit(1)

job()
schedule.every(1).seconds.do(job)
schedule.every().hour.do(job)
schedule.every().day.do(job)

while 1:
    schedule.run_pending()
    time.sleep(1)
```

Demo


My Cluster
3 DBS
12 COLLECTIONS
filter
POCDB
data
ledger
local
13.56.82.17:27017
STANDALONE
MongoDB 3.2.19 Community
data.ledger
DOCUMENTS 934
TOTAL SIZE 140.2KB
AVG. SIZE 154B
INDEXES 1
TOTAL SIZE 60.0KB
AVG. SIZE 60.0KB
Documents
Schema
Explain Plan
Indexes
Validation
FILTER {operation: "insert"}
OPTIONS
ANALYZE
RESET
Query returned 104 documents. This report is based on a sample of 104 documents (100.00%).
microsecs_running
long
min: 51
max: 8679
namespace
string
operation
string
insert
opid
int32
min: 20411
max: 1593131
time
date
S M T W T F S
0:00
6:00
12:00
18:00
23:00
first: 2018-03-12 14:35:38
last: 2018-03-13 11:59:56

Is it advisable to implement our solution?

- This solution is advisable assuming space is not a constraint for a client. In a cloud environment, space constraint can be resolved by simply adding more machines (horizontally Scaling).

- Therefore, as a short answer: Yes, it is advisable, however, if adding more machines leads to going over a client's budget, then this solution may not be advisable in that scenario.

Where would it work well and where would it not (tradeoffs)?

- Main trade off will be a memory consumption.
- It is a concise version of typical NoSQL log entries.
- It will work well while performing analysis of the data so that service provider can take action in advance to prevent query delay.

Who would benefit from such a solution and what are the possible areas where it can be applied?

- Ability to monitor data in heavy traffic time (For example: Thanksgiving weekend) and Low Traffic time during the course of an year.
- Customer can choose an appropriate time interval to insert into ledger (For example: every second, 30 seconds, a minute or more).
- Customer can run machine learning algorithms to analyze the performance of each operation type.
- Customer has flexibility to keep or remove ledger data depending on how much space available in their cloud environment.

Problems Faced

- Installing mongoDB on **docker container** .
- Considered using **Mongo Perf**, however, it was not well documented and properly tested as we kept facing errors running the utility.
- There was no documentation or video tutorials for stress testing using **Jmeter for mongoDB**.
- Understanding blockchain technology was time consuming and implementing it in our solution.

Future Work

- Use this data to do analysis using machine learning algorithm to predict or detect query which are running slowly and take precautionary action in advance.

- Add visualization like clustering data , error detection , graphical representation.

Our Takeaways

- Concepts of Blockchain Technology :
  - Ledger
  - HashCode
  - SHA-256
- How Blockchain used in cryptocurrency
- Working with Jmeter, Mongo Perf
- Working with MongoDB and its commands (mongostat, db.server_status(), mongotop etc)
- Working with the above using Amazon EC2 instance

References

1. “*Building Enterprise-Grade Blockchain Databases with MongoDB*” white paper, 2017. https://www.mongodb.com/white-papers
2. A. Stanciu, "*Blockchain Based Distributed Control System for Edge Computing,*" *2017 21st International Conference on Control Systems and Computer Science (CSCS)*, Bucharest, 2017, pp. 667-671.
3. “MongoDB Documentaion ” https://docs.mongodb.com/manual/tutorial/rotate-log-files/
4. “MongoDB Compass” https://www.mongodb.com/products/compass
5. “PyMongo Documentation” https://api.mongodb.com/python/current/
6. “Apache Jmeter” https://jmeter.apache.org/
7. https://docs.aws.amazon.com/quickstart/latest/mongodb/architecture.html

Questions?

Thank you!