NOSQL

Rahul Bansal
7620987578

Introduction

- NoSQL is a non-relational database management systems, different from traditional relational database management systems in some significant ways. It is designed for distributed data stores where very large scale of data storing needs (for example Google or Facebook which collects terabytes of data every day for their users). These type of data storing may not require fixed schema, avoid join operations and typically scale horizontally.

Why NoSQL ?

- In today's time data is becoming easier to access and capture through third parties such as Facebook, Google+ and others.
  - Personal user information
  - Social graphs
  - Geo location data
  - User-generated content and machine logging data are just a few examples where the data has been increasing exponentially.
  - To avail the above service properly, it is required to process huge amount of data.
  - Which SQL databases were never designed. The evolution of NoSql databases is to handle these huge data properly.

Difference between

RDBMS vs NoSQL

- **RDBMS**
- **NoSQL**

- - Stands for Not Only SQL
- - No declarative query language
- - No predefined schema
- - Key-Value pair storage, Column Store, Document Store, Graph databases
- - Eventual consistency rather ACID property
- - Unstructured and unpredictable data
- - CAP Theorem
- - Prioritizes high performance, high availability and scalability

- Structured and organized data
- Structured query language (SQL)
- Data and its relationships are stored in separate tables.
- Data Manipulation Language, Data Definition Language
- Tight Consistency
- BASE Transaction

Brief history of NoSQL

- The term NoSQL was coined by Carlo Strozzi in the year 1998. He used this term to name his Open Source, Light Weight, DataBase which did not have an SQL interface.
- In the early 2009, when last.fm wanted to organize an event on open-source distributed databases, Eric Evans, a Rackspace employee, reused the term to refer databases which are non-relational, distributed, and does not conform to atomicity, consistency, isolation, durability - four obvious features of traditional relational database systems.
- In the same year, the "no:sql(east)" conference held in Atlanta, USA, NoSQL was discussed and debated a lot.
- And then, discussion and practice of NoSQL got a momentum, and NoSQL saw an unprecedented growth.

CAP Theorem (Brewer's Theorem)

- CAP theorem states that there are three basic requirements which exist in a special relation when designing applications for a distributed architecture.
- **Consistency** - This means that the data in the database remains consistent after the execution of an operation. For example after an update operation all clients see the same data.
- **Availability** - This means that the system is always on (service guarantee availability), no downtime.
- **Partition Tolerance** - This means that the system continues to function even the communication among the servers is unreliable, i.e. the servers may be partitioned into multiple groups that cannot communicate with one another.

- In theoretically it is impossible to fulfill all 3 requirements. CAP provides the basic requirements for a distributed system to follow 2 of the 3 requirements. Therefore all the current NoSQL database follow the different combinations of the C, A, P from the CAP theorem. Here is the brief description of three combinations CA, CP, AP :
- **CA** - Single site cluster, therefore all nodes are always in contact. When a partition occurs, the system blocks.

- CP - Some data may not be accessible, but the rest is still consistent/accurate.
- AP - System is still available under partitioning, but some of the data returned may be inaccurate.

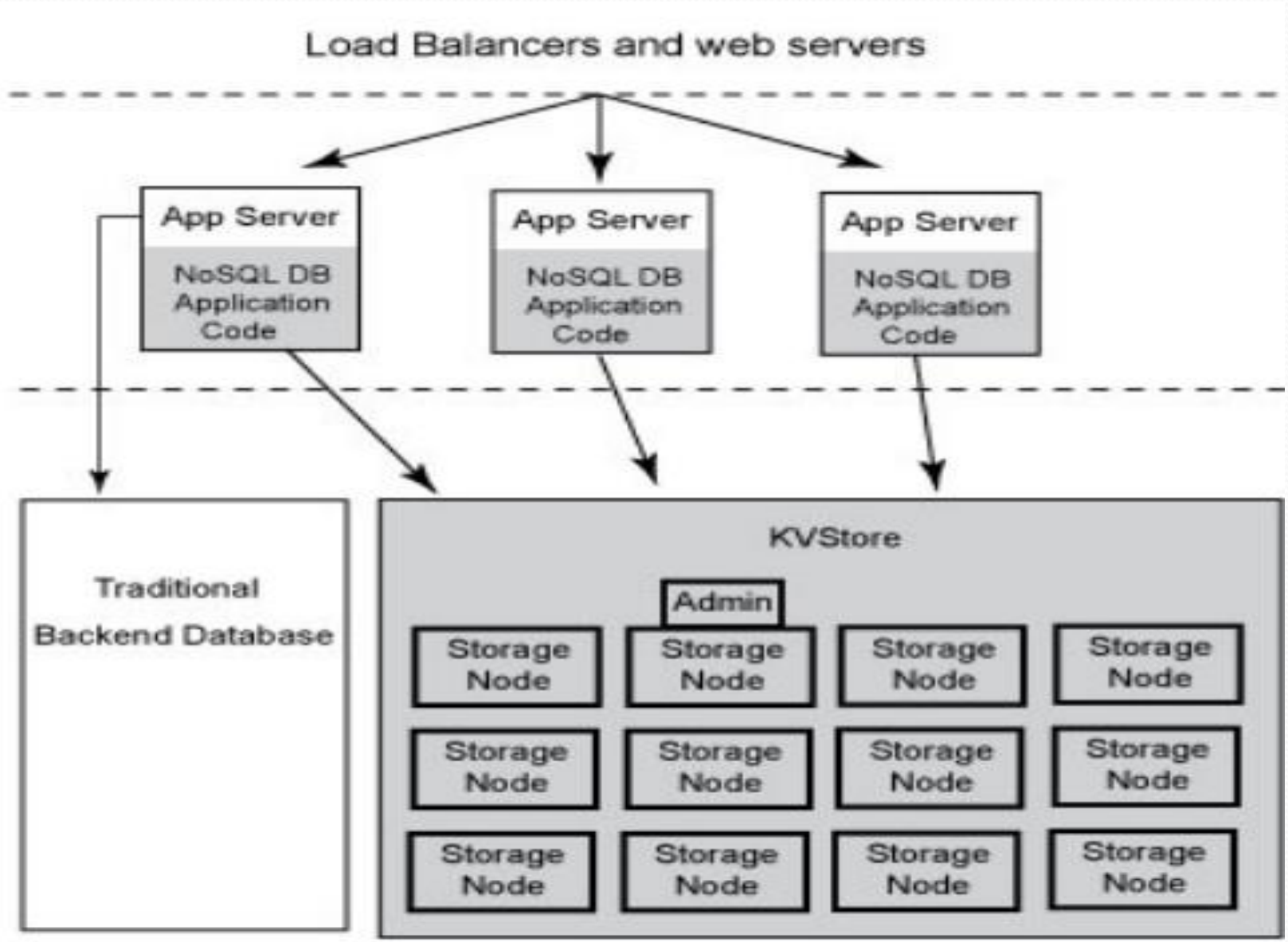
What is MongoDb?

- *MongoDb* is a Open Source database written in C++.
- Drivers and client libraries are typically written in their respective languages, although some drivers use C extensions for better performance.
- If the load increases, by adding more nodes (such as a computer), the performance can be retained.
- It can be used to store data for very high performance applications (for example Foursquare is using it in production).
- MongoDB does not support SQL It supports a rich, ad-hoc query language of its own.
- MongoDb stores data as documents. So it is a *document oriented database*.
- FirstName="Arun", Address="St. Xavier's Road", Spouse=[{Name:"Kiran"}], Children=[{Name:"Rihit", Age:8}].
- Notice there are two different documents (separated by "."). Storing data in this fashion is called as *document oriented database*. *MongoDb* is a document oriented database.
- FirstName="Sameer",Address="8 Gandhi Road".

Architecture Description

- The KVStore is a collection of Storage Nodes which host a set of Replication Nodes.
- A Storage Node is a physical (or virtual) machine with its own local storage.
- A Replication Node can be thought of as a single database which contains key-value pairs
- Every Storage Node hosts one or more Replication Nodes as determined by its capacity, a rough measure of the hardware resources associated with it.

Architecture

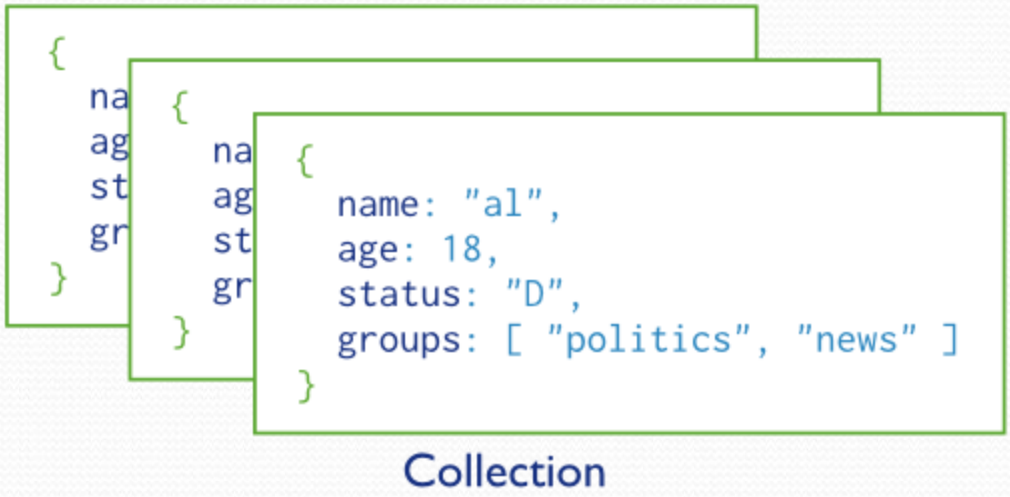


MongoDB CRUD Operations

- MongoDB CRUD Introduction
  - MongoDB stores data in the form of *documents*, which are JSON-like field and value pairs.
  - Documents are analogous to structures in programming languages that associate keys with values (e.g. dictionaries, hashes, maps, and associative arrays).
  - Formally, MongoDB documents are BSON documents.
  - BSON is a binary representation of JSON with additional type information. In the documents, the value of a field can be any of the BSON data types, including other documents, arrays, and arrays of documents.

```
{
  name: "sue",                          <-- field: value
  age: 26,                              <-- field: value
  status: "A",                          <-- field: value
  groups: [ "news", "sports" ]          <-- field: value
}
```

MongoDB stores all documents in collections. A collection is a group of related documents that have a set of shared common indexes. Collections are analogous to a table in relational databases.



Start Working With MongoDB

Installed MongoDB.
Open Command Prompt.
Reach to the MongoDB folder where installed.
cd bin
Create a folder “mongodb\data”
Run mongod.exe --dbpath “d:\set up\mongodb\data”
Open Another Command Prompt.
Reach to the MongoDB folder where installed.
cd bin
mongo.exe
Now You Connected to MongoDB

- Show All DataBases :-
  - show dbs
- Move inside the DataBase
  - Use database name
  - Use student
    - > use student

switched to db student

- Get all the Collections
  - show collections

shows all the collections name

- Create Collection
  - db.createCollection(name);
  - db.createCollection(name,options)
  - db.createCollection(name,

- Get All the data from the collections
  - Select * from studentdata;
  - db.StudentData.find();
- Print the data in Json Format
  - db.StudentData.find().pretty();
- Print Specific Columns
  - Select id,first_name from StudentData;
  - db.StudentData.find({},{id:1,first_name:1});
- Insert Data into the Collection
  - Insert into studentdata values(ffsdfsf);
  - db.StudentData.insert({id:107,first_name:"Suman",last_name:"Aggarwal",age:35,sex:"F",course_id:20,location:"Mumbai",Address:"102"});
- Insert Json inside Json
  - db.StudentData.insert({id:108,first_name:"Suman1",last_name:"Aggarwal1",age:35,sex:"F",course_id:20,location:"Mumbai",Address:{houseno : 101,road:"nagar road",pincode:451295}});

- Insert Json with group of values

```
db.StudentData.insert(
{
    id:109,
first_name:"Suman2",
last_name:"Aggarwal2",
age:35,sex:"F",
course_id:20,
location:"Mumbai",
Address: {
    houseno : 101,
    road:"nagar road",
    pincode:451295
    },
Departmentid : [10,20,30]
}
);
```

- Update the Existing Document :-
  - db.StudentData.update({id:109},{$set : {last_name : "Aggarwal"}});
  - db.StudentData.update({age : {$gt : 34}},{$set : {last_name : "Aggarwal1"}});
  - db.StudentData.update({id:109},{$set : {new_field : “Test"}});
  - db.StudentData.update({"id":110},{$set : {"new_fields":1}})
- Add the New Column to the existing document
  - db.StudentData.update({id: 109},{$set : {new_field:"Test"}},false,true);
  - False and true :: upsert(create new document if not matched) multi(update multiple documents)
- Clear Screen
  - Ctrl + l

- Delete the Document
  - db.StudentData.remove({"_id":ObjectId("56ee300dcf8257bedf8ec7d7“)});
- Drop the Collection
  - db.collection.drop()
- Limit the Rows
  - db.StudentData.find({},{id:1,first_name:1}).limit(3);
- Create Index :: Primary Key
  - db.StudentData.createIndex({ id: 1 }, { unique: true })