An Efficient Design for Multiple Data Stores Cloud Applications

K. Praveen Kumar, Gautam Rampalli

Abstract: ***New data management prerequisites have been released by massive data development and cloud computing. Numerous applications really need to interact on the basis of the types they really need to control with a lot of heterogeneous data stores: standard data types, documents, social network graph data, simple key-value data, etc. A unifying data model is attracting considerable for developers due to flexible communication with heterogeneous relations and NoSQL data stores. The OPEN-PaaS-DataBase API (ODBAPI), which is a well-accepted REST API, allows developers to code their implementation software in the target data storage system separately. Secondly, we propose virtual data stores that interact with the integrated data store wrapped in ODBAPI. It was shown that this prototype responsible for probable solution technologically advanced capable to introduce OpenPaaS circumstances.***

Index Terms: ***NoSQL Data Stores, Relational Data Stores, ODBAPI***

I. INTRODUCTION

Cloud computing has lately become a new computing paradigm that allows for on-demand and extensible resource, platform and software requirement as services. Experts present a set of twenty-one cloud computing definitions [1]. Cloud computing adds execution environments due to its elasticity property for some rapidly developing applications such as large data management. The diverse properties of Big Data are concentrated in the cloud and more accurately on multi-data storage applications. Cloud applications must access and interact with several relations and NoSQL data spaces to fulfil various storage requirements, which have heterogeneous data storage APIs, causing problems while designing, deploying, and migrating multi-data storage applications. A variety of contemporary applications and studies, such as in [2-4] and so forth, take cloud information management into consideration. In addition, they are not sufficient to tackle all cloud data management intentions [5]. As stated earlier Polyglot persistence [6] is called the use of various data stores simultaneously. For instance, an implementation can use a relational data store and a NoSQL data store concurrently or split information into various data stores.

We presented and discussed the minimum standards of this environment in previous work [7] and analyzed the latest state of the art. In [8], the authors identify the issues and the limits of a user's use of NoSQL data stores. This is because a distinctive querying language and API between NoSQL and SQL data storage are not available and optimized. In [9] the writers use JDBC in order to get in touch with multiple classes (i.e. Oracle, MySQL, etc.) of relational DBMS. But interacting with various DBMS classes is harder in the cloud because there are many feasible, heterogeneous data stores on all sides, such as the data model, request language, application systems etc. NoSQL DBMS [10] are widely used and not uniform in cloud as they contain a distinct data model, patented language and stability designs depending on possible accuracy.

In [8] the writers show the problems and limitations of a customer in NoSQL data stores. The issues arise because optimization is not available and because there is no particular query language and API between NoSQL data stores and NoSQL data stores. In [9], writers use JDBC to communicate with uncommon kinds of relational DBMS for applications such as Oracle, MySQL, and so forth. It's however more complicated to interact with multiple kinds of DBMS in the web, as there are many needed information shops that are quite different in all dimensions: information system, request vocabulary, application template, etc. NoSQL DBMS [10] is widely used in cloud applications and is completely unstandardized as its information model differs from its proprietary query language, and reliability measures are applied with a necessary accuracy.

The Spring Data Framework [11] provides certain general concepts for particular DBMS NoSQL kinds and associated DBMS. These abstractions are advanced for each DBMS. It focus them as credible programming model relies on a collection of abstractions and designs of the Spring Framework. It's not so simple to add a new data warehouse. But the answer is strongly related to the Java model of programming. The writers suggest a prevalent [12] runtime framework for seamless connection to NoSQL and associated Save Our Systems (SOS) data warehouses. SOS is a database entry element with such an implementation and the different data stores it utilizes.

In practice to do this, writers have established a popular feature to reach particular NoSQL DBMSs and a mutual data model to link apps to the destination data stores. They claim that SOS can be used to incorporate the data storage connection; there is no proof of an effective and extensible scheme. Object-NoSQL Data store Mapper (ONDM) [13] is a structure to enable the processing and recovery in NoSQL data stores of persistent objects. It effectively supplied a NoSQL-based apps developer with an Object Relational Mapping-like (ORM-like) API (e.g., JPA API). ONDM does not, however, suggest storing related information. The writers introduce the notion of a REST-based API in [14]. This API allows you to communicate with multiple DBMs (DaaSs), be it relation-based DBMS or NoSQL DBMS. They suggest fresh terminology for distinct

Revised Manuscript Received on July 22, 2019.

Dr. K. Praveen Kumar, Working as Assistant Professor, Department of IT, KITS, Warangal, Telangana, India.

Gautam Rampalli, Assistant Professor, Department of IT, KITS, Warangal, Telangana, India

ideas for each sort of DBMS. From then on, the software communicates in heterogeneous data stores with different APIs. Therefore, the designer needs to know a big amount of APIs. In order to tackle this issue, we suggest a versatile and consistent REST API in this paper, which permits CRUD activities in separate NoSQL and relational databases. This API is called ODBAPI and has the dual purpose: firstly, it simplifies the job of the designer when moving from one database to another. A single API makes it simpler to modify an application's source code. Secondly, it decreases the strain of communicating simultaneously with various data stores using ODBAPI.

II. BACKGROUND WORK

OpenPaaS is an arrangement that allows applications to be built and implemented depending on demonstrated technology provided by suppliers such as cooperative messaging technology, inclusion and workflow technology in order to satisfy cloud computing demands (see Figure.1).

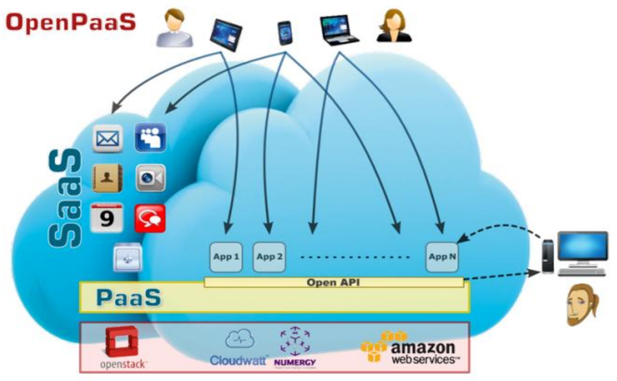


Figure 1. OpenPaaS overview

A. Migration of application from one cloud to another

Usually, the cloud ecosystem offers single data storage for executed applications. In addition, in certain conditions, this data storage model may not comply with all implementation criteria. An application then requires to move from one data store to another in order to generate a mildly more useful data store. Please note that applications can move to another storage setting for new demands in the data-store. In Figure.2, we demonstrate an implementation migration situation as *A* must shift from one CouchDB cloud environment to the next to meet potential data needs. The application connects data to a cloud-based environment from another Mongo DB file store.

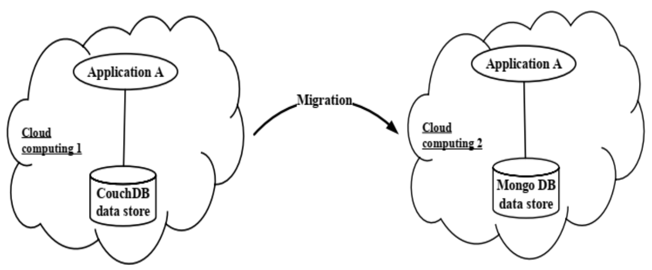


Figure 2. Migration of application from one cloud to another

As discussed earlier migration of application is very simple and robust to the first quick glance; however, the work is cumbersome and diligent behind all this situation. Indeed, the software source code must be re-adopted to interact with the fresh data store. To do so, necessity of figure out and be acquainted with his proprietary API. Data must also be moved from the old to the new data store.

B. Second scenario: Polyglot persistence

It follows from the fact that a request can use multiple data stores that suit what is commonly regarded as polyglot storage in a virtual world as depicted particular instance in Figure 3. Application *A* conveys with three heterogeneous data stores, a relationship data store, a data store for CouchDB documents, and Riak, the main data store.

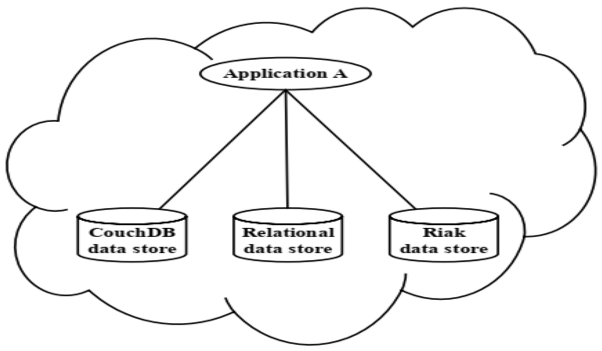


Figure 3. Multiple data store use in one cloud environment

If the framework requires to search data from multiple data stores (e.g., connection data, aggregating data, etc.) it cannot do so compostable without any kind of mediation.

III. PROPOSED MODEL

The approach of a system mainly relies on the four elements as depicted in figure.4. Those are as follows:

- Data Model
- REST API/Services
- Virtual Data Stores
- Dedicated Components for Discovery and Deployment

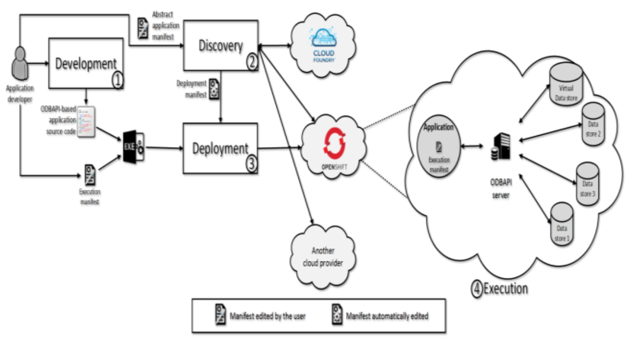


Figure 4. Summary of our elucidation model

A. Unifying Data Model

We define a data model that illustrates the integrated data collection models (explicit/implicit) and provides a common and unified view in order to classify queries through heterogeneous data storage models. In the process of development, designers have a global data model that is conveyed in accordance with our Unification model and that integrates local data, storage models.

B. REST API/Services

By unifying data models we identify a useful model of resources with a REST API called the ODBAPI so that we can interact in a truly unique and uniform way with the participating data stores. The ODBAPI enforcement provider REST can then enclose each data storage. Our API adjusts the relations between your specific drivers and statistical stores dynamically.

C. Virtual Data Stores:

In our paradigm, virtual data storage determines a particular element accountable for queries transmitted through a data storage application. A Virtual Data Store (VDS) (i) is available as an ODBAPI Resting Service (ii) in conjunction with our Unifying Registration Model and a set of communications instructions, and (ii) the wrapper rest service endpoints can proceed with that model throughout the world.

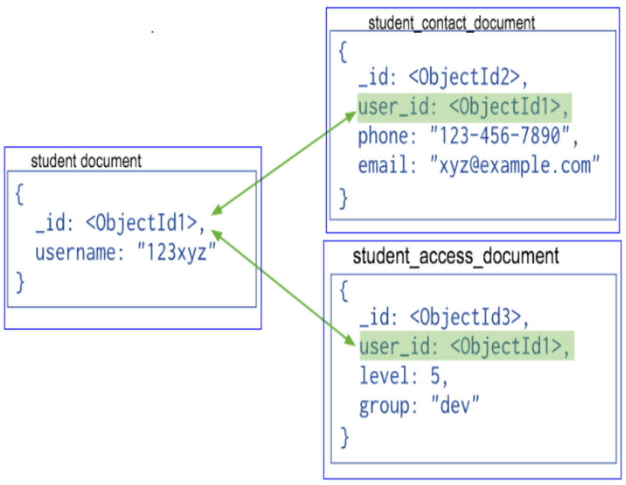
D. Dedicated components for discovery and deployment

In our mechanism, we categorize two components, discovery and deployment modules, which are capable of identifying and implementing suitable cloud environments. Developers first show their requirements in the data stores and the computer ecosystem via a theoretical application manifest as shown in Fig.4. On the basis of this manifest, the discovery component perceives and decides the appropriate cloud environment and generates a manifest offer. This manifest will be used by the deployment component to deploy the application in that specific context.

E. Open PAAS Database API

In this section, we provide an ODBAPI that is a resting API that enables CRUD operations to be carried out in different types of data stores to support our unifying model. This API offers the design level and a constant communication with cloud-based data stores.

F. Document Databases: A document database's concentrate phenomenon is "document."



student document
```
{
  _id: <ObjectId1>,
  username: "123xyz"
}
```

student_contact_document
```
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "123-456-7890",
  email: "xyz@example.com"
}
```

student_access_document
```
{
  _id: <ObjectId3>,
  user_id: <ObjectId1>,
  level: 5,
  group: "dev"
}
```

Figure 5: Normalized Document Model

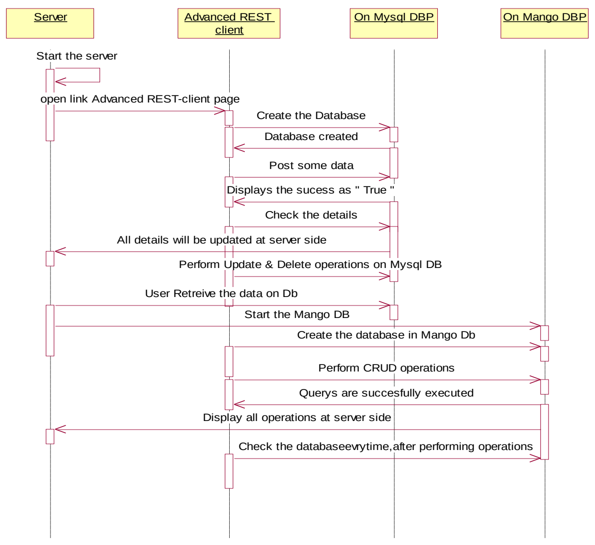

Server
Advanced REST client
On Mysql DBP
On Mango DBP
Start the server
open link Advanced REST-client page
Create the Database
Database created
Post some data
Displays the sucess as " True "
Check the details
All details will be updated at server side
Perform Update & Delete operations on Mysql DB
User Retreive the data on Db
Start the Mango DB
Create the database in Mango Db
Perform CRUD operations
Querys are succesfully executed
Display all operations at server side
Check the databaseevrytime,after performing operations

Figure 6. Sequence diagram

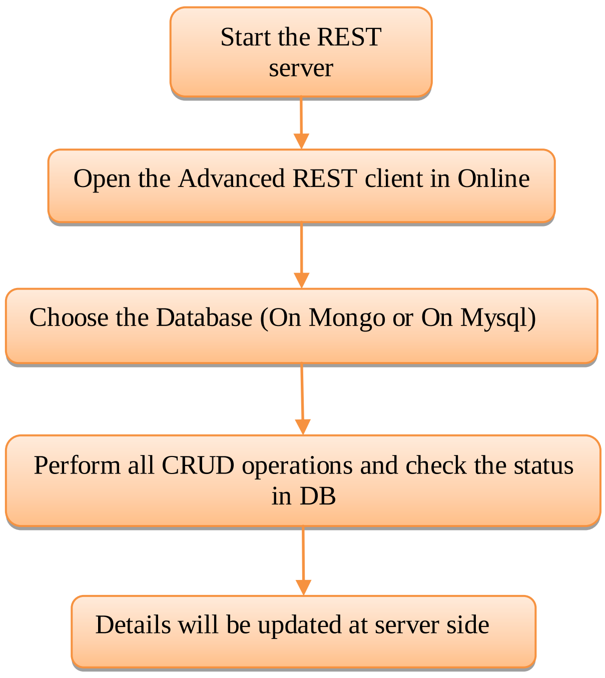
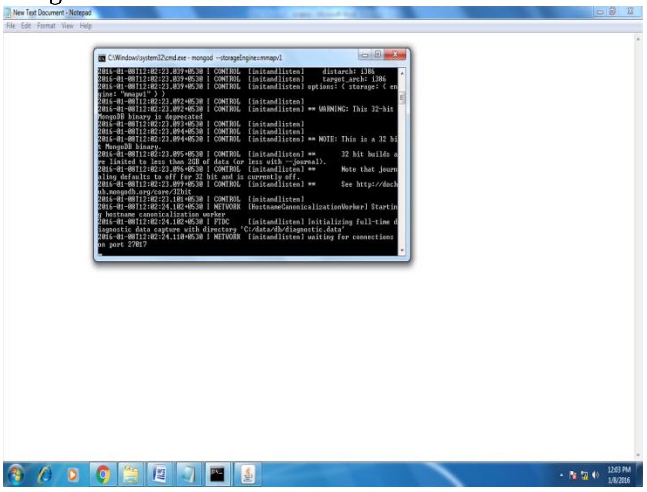

Start the REST server
Open the Advanced REST client in Online
Choose the Database (On Mongo or On Mysql)
Perform all CRUD operations and check the status in DB
Details will be updated at server side

Figure. 7: Flow process of Database update environment

Every document - driven construction of the store varies from the data in its concept, all of which are founded upon the logic that ' documents encapsulate and enraptures data (or data) in standardization encoding or formats, but most popular encoding uses YAML, XML, JSON and BSON binary forms. Documentation is closely linked to a distinctive primary key, using a key for this document in the store. One essential feature of a document-oriented store is that the store offers a query language or API that gathers documents based on its contents, for quick search of a key-value database. The classic example of the standardized document model is provided in Figure.5.

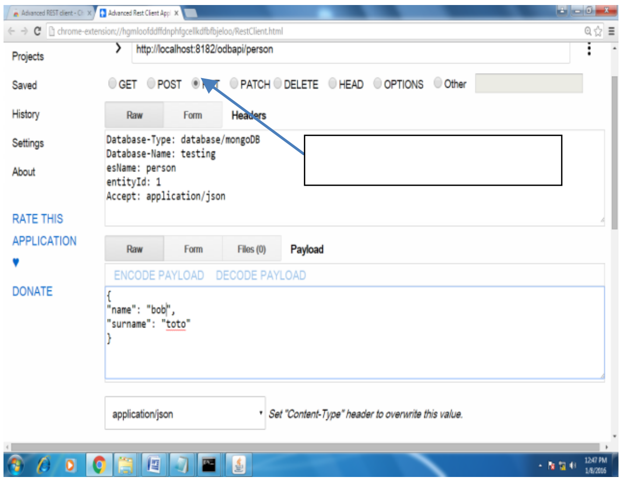
IV. RESULTS & DISCUSSION

To show the feasibility and usefulness of our API, we provide a client we call an ODBAPI client. The latter allows a developer to use JAVA techniques for ODBAPI operation. It is therefore easy for him to schedule his application. We also created the next ODBAPI mainly based client to manage relationships and NoSQL data stores in a cloud company. We show a screen capture of this customer's consumer interface. It actually outlines two heterogeneous MYSQL and MongoDB information stores.



```
2016-01-08T12:02:23.039+0530 I CONTROL  [initandlisten]     distarch: i386
2016-01-08T12:02:23.039+0530 I CONTROL  [initandlisten]     target_arch: i386
2016-01-08T12:02:23.039+0530 I CONTROL  [initandlisten] options: { storage: { en
gine: "mmapv1" } }
2016-01-08T12:02:23.092+0530 I CONTROL  [initandlisten]
2016-01-08T12:02:23.092+0530 I CONTROL  [initandlisten] ** WARNING: This 32-bit
MongoDB binary is deprecated
2016-01-08T12:02:23.093+0530 I CONTROL  [initandlisten]
2016-01-08T12:02:23.094+0530 I CONTROL  [initandlisten]
2016-01-08T12:02:23.094+0530 I CONTROL  [initandlisten] ** NOTE: This is a 32 bi
t MongoDB binary.
2016-01-08T12:02:23.095+0530 I CONTROL  [initandlisten] **       32 bit builds a
re limited to less than 2GB of data (or less with --journal).
2016-01-08T12:02:23.096+0530 I CONTROL  [initandlisten] **       Note that journ
aling defaults to off for 32 bit and is currently off.
2016-01-08T12:02:23.099+0530 I CONTROL  [initandlisten] **       See http://doch
ub.mongodb.org/core/32bit
2016-01-08T12:02:23.101+0530 I CONTROL  [initandlisten]
2016-01-08T12:02:24.102+0530 I NETWORK  [HostnameCanonicalizationWorker] Startin
g hostname canonicalization worker
2016-01-08T12:02:24.102+0530 I FTDC     [initandlisten] Initializing full-time d
iagnostic data capture with directory 'C:/data/db/diagnostic.data'
2016-01-08T12:02:24.110+0530 I NETWORK  [initandlisten] waiting for connections
on port 27017
```

Figure. 8: starting the MongoDB



http://localhost:8182/odbapi/person

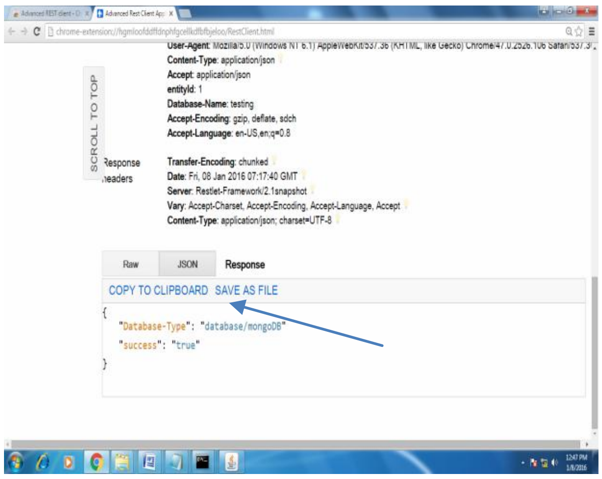
GET POST [✓ PUT] PATCH DELETE HEAD OPTIONS Other

```
Database-Type: database/mongoDB
Database-Name: testing
esName: person
entityId: 1
Accept: application/json
```

```
{
"name": "bob",
"surname": "toto"
}
```

application/json — Set "Content-Type" header to overwrite this value.

Figure. 9: Performing put operation on Database

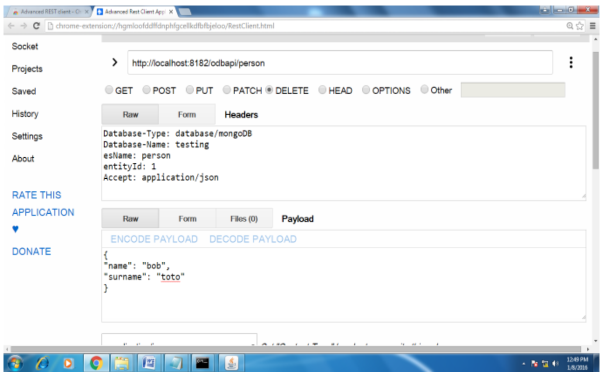


User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.3
Content-Type: application/json
Accept: application/json
entityId: 1
Database-Name: testing
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8

Transfer-Encoding: chunked
Date: Fri, 08 Jan 2016 07:17:40 GMT
Server: Restlet-Framework/2.1snapshot
Vary: Accept-Charset, Accept-Encoding, Accept-Language, Accept
Content-Type: application/json; charset=UTF-8

```
{
  "Database-Type": "database/mongoDB"
  "success": "true"
}
```

Figure. 10: PUT operation successfully updated



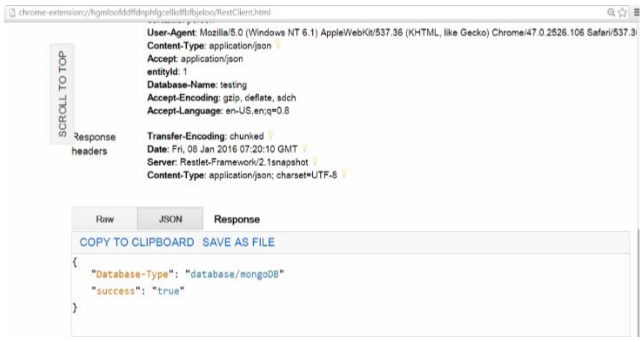
http://localhost:8182/odbapi/person

GET POST PUT PATCH [✓ DELETE] HEAD OPTIONS Other

```
Database-Type: database/mongoDB
Database-Name: testing
esName: person
entityId: 1
Accept: application/json
```

```
{
"name": "bob",
"surname": "toto"
}
```

Figure. 11: Searching for a record from MongoDB

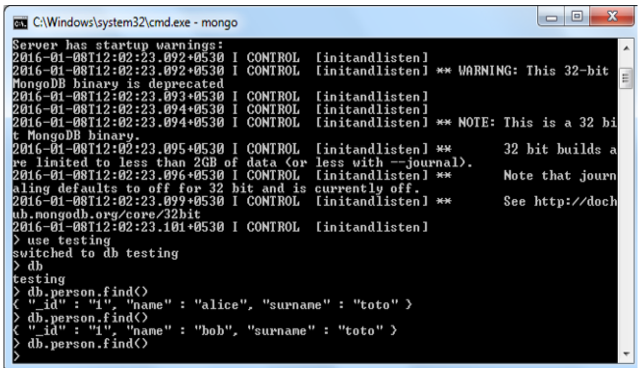


User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.3
Content-Type: application/json
Accept: application/json
entityId: 1
Database-Name: testing
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8

Transfer-Encoding: chunked
Date: Fri, 08 Jan 2016 07:20:10 GMT
Server: Restlet-Framework/2.1snapshot
Content-Type: application/json; charset=UTF-8

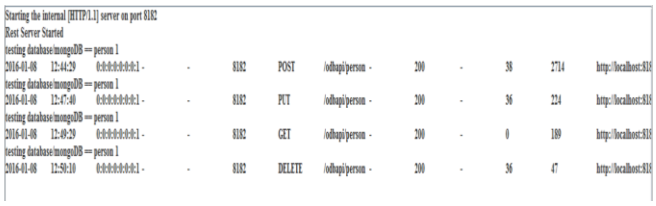
```
{
  "Database-Type": "database/mongoDB"
  "success": "true"
}
```

Figure. 12: Deleting a record from DB



```
Server has startup warnings:
2016-01-08T12:02:23.092+0530 I CONTROL  [initandlisten]
2016-01-08T12:02:23.092+0530 I CONTROL  [initandlisten] ** WARNING: This 32-bit
MongoDB binary is deprecated
2016-01-08T12:02:23.093+0530 I CONTROL  [initandlisten]
2016-01-08T12:02:23.094+0530 I CONTROL  [initandlisten]
2016-01-08T12:02:23.094+0530 I CONTROL  [initandlisten] ** NOTE: This is a 32 bi
t MongoDB binary.
2016-01-08T12:02:23.095+0530 I CONTROL  [initandlisten] **       32 bit builds a
re limited to less than 2GB of data (or less with --journal).
2016-01-08T12:02:23.096+0530 I CONTROL  [initandlisten] **       Note that journ
aling defaults to off for 32 bit and is currently off.
2016-01-08T12:02:23.099+0530 I CONTROL  [initandlisten] **       See http://doch
ub.mongodb.org/core/32bit
2016-01-08T12:02:23.101+0530 I CONTROL  [initandlisten]
> use testing
switched to db testing
> db
testing
> db.person.find()
{ "_id" : "1", "name" : "alice", "surname" : "toto" }
> db.person.find()
{ "_id" : "1", "name" : "bob", "surname" : "toto" }
> db.person.find()
>
```

Figure. 13: Check in the database



```
Starting the internal [HTTP/1.1] server on port 8182
Rest Server Started
testing database/mongoDB == person 1
2016-01-08   12:44:29   0:0:0:0:0:0:0:1 -    -    8182   POST     /odbapi/person -   200   -   38   2714   http://localhost:818
testing database/mongoDB == person 1
2016-01-08   12:47:40   0:0:0:0:0:0:0:1 -    -    8182   PUT      /odbapi/person -   200   -   36   224    http://localhost:818
testing database/mongoDB == person 1
2016-01-08   12:49:29   0:0:0:0:0:0:0:1 -    -    8182   GET      /odbapi/person -   200   -   0    189    http://localhost:818
testing database/mongoDB == person 1
2016-01-08   12:50:10   0:0:0:0:0:0:0:1 -    -    8182   DELETE   /odbapi/person -   200   -   36   47     http://localhost:818
```

Figure. 14: Final Server screen

V. CONCLUSION

We store different types of data in different data stores. Each application needs certain types of data to be stored and manipulated in a number of heterogeneous data stores. We introduce an integrated data model incorporated in the REST ODBAPI, which allows for a fully uniform and uniform interaction with the data sources involved. Our approach to detect and implement an application in a cloud environment with great performance of complex evaluation and execution can handle virtual data sources.

REFERENCES

1. C. Baun, M. Kunze, J. Nimis, and S. Tai, Cloud Computing - Web-Based Dynamic IT Services. Springer, 2011.
2. C. Fay and et al., “Bigtable: A distributed storage system for structured data,” ACM Trans. Comput. Syst., vol. 26, no. 2, 2008.
3. B. F. Cooper and et al., “Pnuts: Yahoo!’s hosted data serving platform,” PVLDB, vol. 1, no. 2, pp. 1277–1288, 2008.
4. G. DeCandia and et al., “Dynamo: amazon’s highly available key-value store,” in Proceedings of the 21st ACM Symposium on Operating Systems Principles, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007, pp. 205–220.
5. D. Agrawal, A. El Abbadi, S. Das, and A. J. Elmore, “Database scalability, elasticity, and autonomy in the cloud - (extended abstract),” in Database Systems for Advanced Applications - 16th International Conference, DASFAA 2011, Hong Kong, China, April 22-25, 2011, Proceedings, Part I, 2011, pp. 2–15.
6. P. Sadalage and M. Fowler, NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence, ser. always learning. Addison Wesley Professional, 2012.
7. R. Sellami and B. Defude, “Using multiple data stores in the cloud: Challenges and solutions,” in Data Management in Cloud, Grid and P2P Systems - 6th International Conference, Globe 2013, Prague, Czech Republic, August 28-29, 2013. Proceedings, 2013, pp. 87–98.
8. M. Stonebraker, “Stonebraker on nosql and enterprises,” Commun. ACM, vol. 54, no. 8, pp. 10–11, 2011.
9. M. Fisher, J. Ellis, and J. C. Bruce, JDBC API Tutorial and Reference, 3rd ed. Pearson Education, 2003.
10. A. B. M. Moniruzzaman and S. A. Hossain, “Nosql database: New era of databases for big data analytics - classification, characteristics and comparison,” CoRR, vol. abs/1307.0191, 2013.
11. M. Pollack, O. Gierke, T. Risberg, J. Brisbin, and M. Hunger, Eds., Spring Data. O’Reilly Media, October 2012.
12. P. Atzeni, F. Bugiotti, and L. Rossi, “Uniform access to nonrelational database systems: The SOS platform,” in Advanced Information Systems Engineering - 24th International Conference, CAiSE 2012, Gdansk, Poland, June 25-29, 2012. Proceedings, 2012, pp. 160–174.
13. L. Cabibbo, “Ondm: an object-NOSQL data store mapper,” Faculty of Engineering, Roma Tre University. Retrieved June 15th, 2013.
14. T. Haselmann, G. Thies, and G. Vossen, “Looking into a rest based universal API for database-as-a-service systems,” in 12th IEEE Conference on Commerce and Enterprise Computing, CEC 2010, Shanghai, China, November 10-12, 2010, 2010, pp. 17–24.

AUTHORS PROFILE

Dr. K. Praveen Kumar received PhD in Computer Science & Engineering in 2015, M.Tech in Software Engineering from Kakatiya Institute of Technology & Science Warangal, Telangana, India in 2010 and B.Tech in Information Technology from Kakatiya Institute of Technology & Science Warangal, Telangana, India 2007. Presently working as Assistant Professor, in Information Technology Department, Kakatiya Institute of Technology and Science, Warangal, Telangana. India. His Research interests are Cloud Computing, IoT. He has published more than 20 research papers in reputed Journals.

Mr. Gautam Rampalli is an Assistant Professor in the Department of Information Technology at Kakatiya Institute of Technology & Science, Warangal. He has 8 years of work experience in the areas of teaching, administrative, programming, student’s affairs and organizing seminars, workshops, technical events. He obtained his Master of Technology degree in Software Engineering and Bachelor of Technology in Computer Science & Engineering. He has authored and coauthored 9 publications. He is a reviewer committee board member for International Journal of Science and Research (IJSR). His current field placement is in “Cloud Computing”. He has also been engaged to create linkage between industry and academia. He is a life member in Indian Society for Technical Education.