SUPPORTING MULTI DATA STORES APPLICATIONS IN CLOUD ENVIRONMENTS

KALLAM PRASANNA, T.SWARNALATHA
[1]M.Tech Student in CSE, NALANDA INSTITUTE OF ENGINEERING TECHNOLOGY, AP
[2]Associate Professor, Dept of CSE, NALANDA INSTITUTE OF ENGINEERING TECHNOLOGY, AP

ABSTRACT: The production of huge amount of data and the emergence of cloud computing have introduced new requirements for data management. Many applications need to interact with several heterogeneous data stores depending on the type of data they have to manage: traditional data types, documents, graph data from social networks, simple key-value data, etc. Interacting with heterogeneous data models via different APIs, and multiple data store applications imposes challenging tasks to their developers. Indeed, programmers have to be familiar with different APIs. In addition, the execution of complex queries over heterogeneous data models cannot, currently, be achieved in a declarative way as it is used to be with mono-data store application, and therefore requires extra implementation efforts. Moreover, developers need to master and deal with the complex processes of cloud discovery, and application deployment and execution. In this paper we propose an integrated set of models, algorithms and tools aiming at alleviating developers task for developing, deploying and migrating multiple data stores applications in cloud environments. Our approach focuses mainly on three points. First, we provide a unifying data model used by applications developers to interact with heterogeneous relational and NoSQL data stores. Based on that, they express queries using OPEN-PaaS-Data Base API (ODBAPI), a unique REST API allowing programmer to write their applications code independently of the target data stores. Second, we propose virtual data stores, which act as a mediator and interact with integrated data stores wrapped by ODBAPI. This run-time component supports the execution of single and complex queries over heterogeneous data stores. Finally, we present a declarative approach that enables to lighten the burden of the tedious and non-standard tasks of (1) discovering relevant cloud environment and (2) deploying applications on them while letting developers to simply focus on specifying their storage and computing requirements. A prototype of the proposed solution has been developed and is currently used to implement use cases from the OpenPaaS project.

Index Terms— REST-based API, NoSQL data stores, relational data stores, join queries, polyglot persistence, manifest based matching.

I. INTRODUCTION

Cloud computing has recently emerged as a new computing paradigm enabling on-demand and scalable provision of resources, platforms and software as services. Cloud computing is often presented at three levels : the Infrastructure as a Service (IaaS) giving access to abstracted view on the hardware, the Platform-as-a-Service (PaaS) providing programming and execution environments to the developers, and the Sofwtare as a Service (SaaS) enabling software applications to be used by cloud's end users. Due to its elasticity property, cloud computing provides interesting execution environments for several emerging applications such as big data management. According to the National Institute of Standards and Technology1 (NIST), big data is *data which exceed the capacity or capability of current or conventional methods and systems*. It is mainly based on the 3-Vs model where the three Vs refer to volume, velocity and variety properties [2]. Volume means the processing of large amounts of information. Velocity signifies the increasing rate at which data flows. Finally, variety refers to the diversity of data sources. Several people have also proposed to add more V to this definition. Veracity is widely proposed and represents the quality of data (accuracy, freshness, consistency etc.). Against this background, the challenges of big data management result from the expansion of the 3Vs properties. In our work, we focus mainly on the variety property and more precisely on multiple data store based applications in the cloud. In order to satisfy different storage requirements, cloud applications usually need to access and interact with different relational and NoSQL data stores having heterogeneous APIs. The heterogeneity of the data stores induces several problems when developing, deploying and migrating multiple data store applications. Below, we list the main four problems which we are tackling in this paper. *Pb*1 Heavy workload on the application developer: Nowadays data stores have different and heterogeneous APIs. Developers of multiple data store based applications need to be familiar with all these APIs when coding their applications. *Pb*2 No declarative way for executing complex queries ue to the heterogeneity of the data models, there is currently no declarative way to define and execute complex queries over several data stores. This is mainly due to the absence of a global schema of heterogeneous data stores. In addition, NoSQL data stores are schemeless. That means developers have to cope themselves with the implementation of such queries. *Pb*3 Code adaptation: When migrating applications from one cloud environment to another, application devel opers have to re-adapt the application source code in order to interact with new data stores. Developers have potentially to learn and master new APIs. *Pb*4 Tedious and non-standard processes of discovery and deployment: Once an application is developed or migrated, developers have to deploy it into a cloud provider. Discovering the most suitable cloud environment providing the required data stores and deploying the application on it are tedious and meticulous provider-specific process. Consistency is also an important issue in multi datastores applications. In fact, cloud data stores in general implement different consistency models (strong consistency model for RDBMS and weak consistency models for NoSQL DBMS). This implies that the consistency model at the application level is not really defined. We do not address this issue in this paper, focusing only on querying. The interested reader may read [3] which proposes a middleware service addressing the problem of client-centric consistency on top of eventually consistent distributed data stores (Amazon S3 for example). In this paper we propose an integrated set of models, algorithms and tools aiming at alleviating developers' tasks for developing, deploying and migrating multiple data stores based applications in cloud environment.

II. EXISISTING SYSTEM

Cloud computing provides interesting execution environments for several emerging applications such as big data management. According to the National Institute of Standards and Technology1 (NIST), big data is data which exceed the capacity or capability of current or conventional methods and systems. It is mainly based on the 3-Vs model where the three Vs refer to volume, velocity and variety properties. Volume means the processing of large amounts of information. Velocity signifies the increasing rate at which data flows. Finally, variety refers to the diversity of data sources. Several people have also proposed to add more V to this definition. Veracity is widely proposed and represents the quality of data.

Disadvantages:

- We focus mainly on the variety property and more precisely on multiple data store based applications in the cloud.
- Cloud applications usually need to access and interact with different relational and NoSQL data stores having heterogeneous APIs. The heterogeneity of the data stores induces several problems when developing, deploying and migrating multiple data store applications.
- Heavy workload on the application developer.
- No declarative way for executing complex queries.
- Code adaptation: When migrating applications from one cloud environment to another, application developers have to re-adapt the application source code in order to interact with new data stores.
- Tedious and non-standard processes of discovery and deployment.

III. PROPOSED SYSTEM

We propose an integrated set of models, algorithms and tools aiming at alleviating developers' tasks for developing, deploying and migrating multiple data stores based applications in cloud environment. First, we define a unifying data model used by applications developers to interact with different data stores. This model tackles the problem of heterogeneity between data models and the absence of schemes in NoSQL data stores. Second, we propose virtual data stores (VDS) to evaluate and optimize the execution of queries - especially complex ones- over different data stores (see section 6). In order to support the definition and the execution of queries over heterogeneous data models, we use the unifying data model that we accomplish with correspondence rules. Third, we present a declarative approach for discovering appropriate cloud environments and deploying applications on them while letting developers simply focus on specifying their storage and computing requirements.

Advantages:

- Using unifying data model developers may express and execute any type of queries using OPEN-PaaS-DataBase API (ODBAPI).
- The highlights of ODBAPI are twofold: (i) decoupling cloud applications from data stores in order to facilitate their development and their migration, and
- (ii) Easing the developer's task by lightening the burden of managing different APIs.
- Virtual data stores for complex queries execution.

- Manifest for data stores discovery and automatic application deployment.

Architecture:

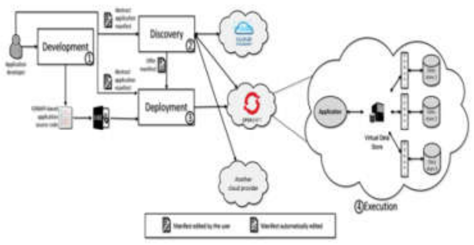


Fig. 1: Overview of our solution

MODULES:

1. Unifying data model
2. REST API/services
3. Virtual data stores

Modules Description:

Unifying data model:

We define a data model which abstracts from the underlying (explicit/implicit) integrated data store models, and provide a common and unified view so that developers can define their queries over heterogeneous data stores. During the development step, the developers dispose of a global data model expressed according to our unifying model and which integrates local data store models. Our unifying data model decouples query definitions from the data stores specific languages.

REST API/services:

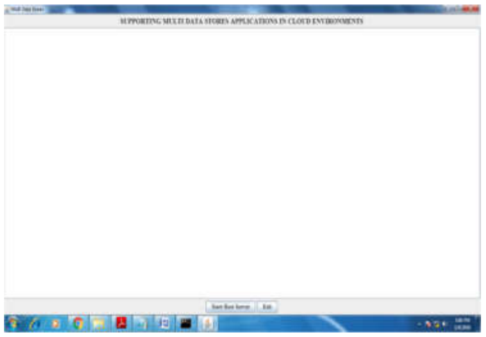
Based on our unifying data model, we define a resource model upon which we develop a REST API, called ODBAPI, enabling to interact with involved data stores in a unique and uniform way. Each data store will be then wrapped behind a REST service implementing ODBAPI. Our API decouples the interactions with data stores from their specific drivers. By using our unifying data model to express the queries and ODBAPI to interact with the data stores, developers do not have to deal with various languages and APIs and do not have to adapt their code.

Virtual data stores:

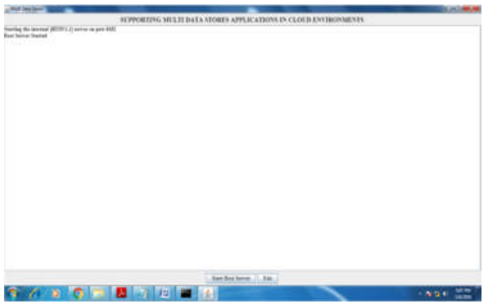
Wrapper REST services enable executing simple queries over the involved data stores. However, they are not meant to execute complex queries (such as join, union, etc.). In our approach, we consider virtual data store (VDS for short) a specific component responsible for executing queries submitted by a multiple data store application. A VDS holds the global data model integrating the different data stores and which is specified according to our unifying data model and a set of correspondence rules.

IV. SCREEN SHOTS

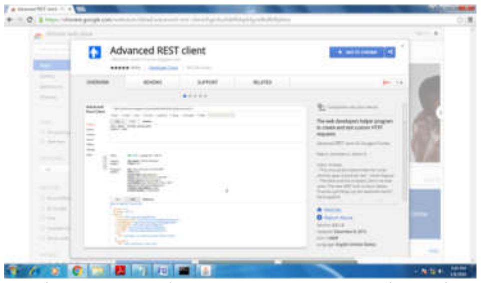
Welcome screen:


SUPPORTING MULTI DATA STORES APPLICATIONS IN CLOUD ENVIRONMENTS

Click on start rest server:


SUPPORTING MULTI DATA STORES APPLICATIONS IN CLOUD ENVIRONMENTS

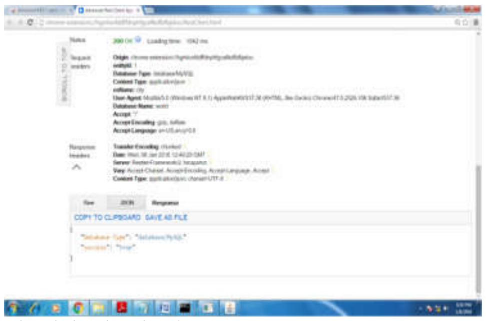
Rest for chrome:



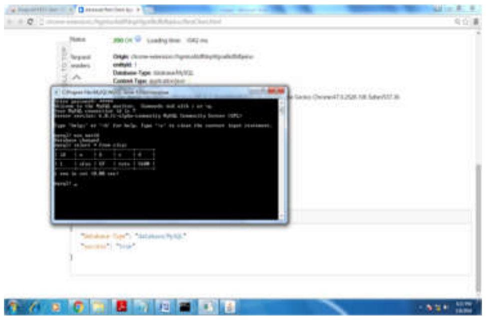
Posting some data on MY SQL (inserting the records)

Here we are connecting to MYSQL, db name in world and table name is city.

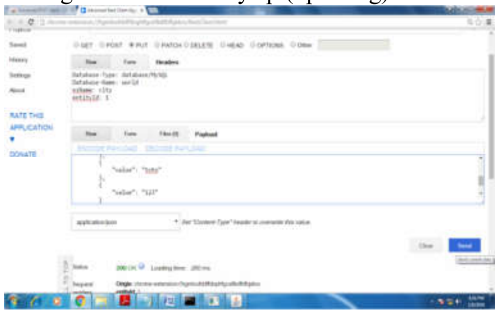
After successfully posting the data:



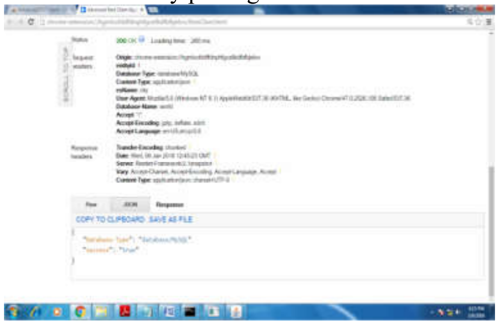
Check in the database:



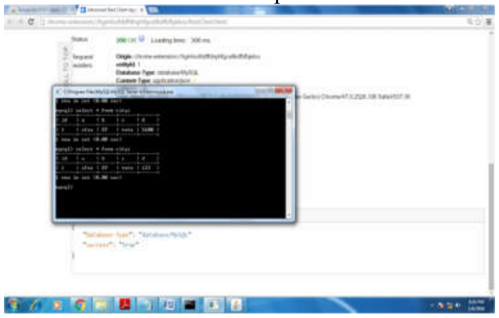
Putting some data on mysql (updating):



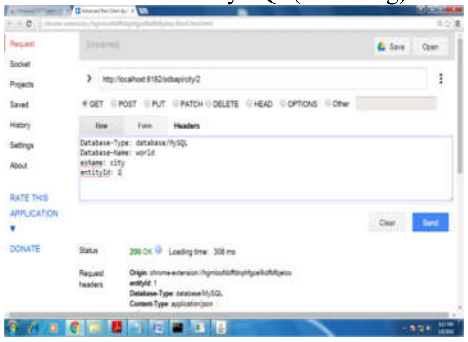
After successfully putting the data:



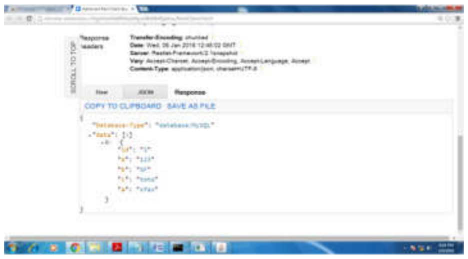
Check in database for the updated value:



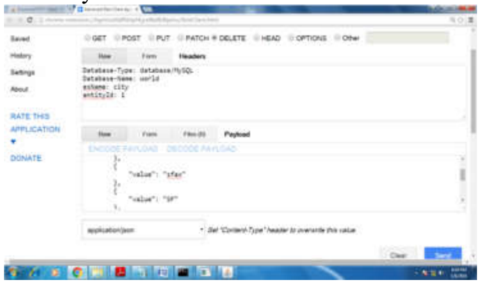
Get some data from My SQL(retrieving)



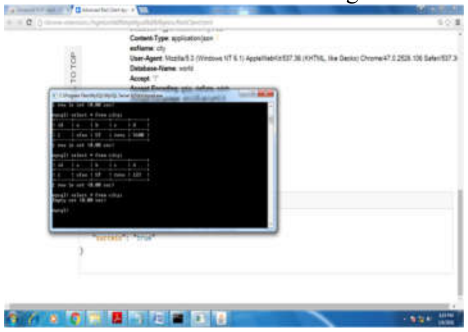
Retrived information from the database:



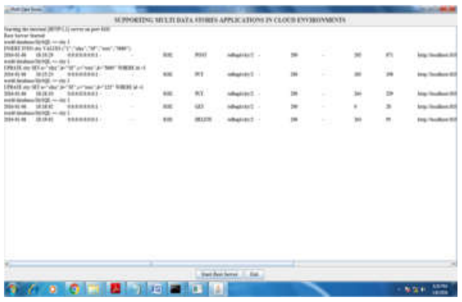
Similarly we can delete the records:



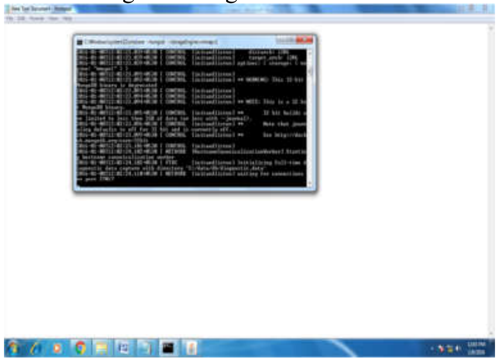
Check in the database after deleting:



Server screen:

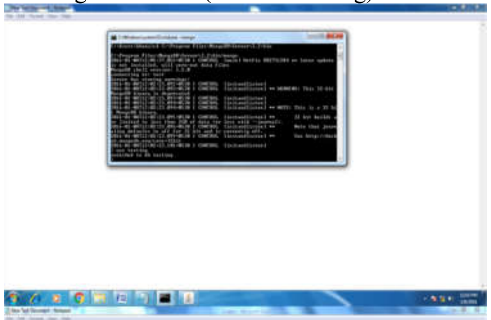


After starting the Mongo db:

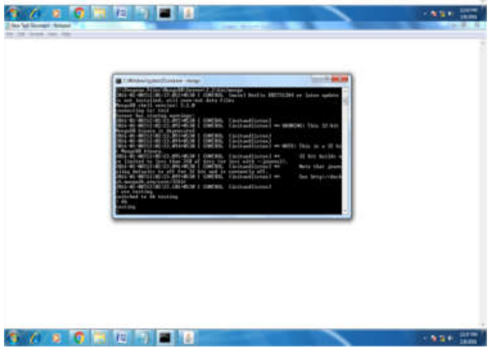


[illegible]

Creating a database (name as testing)

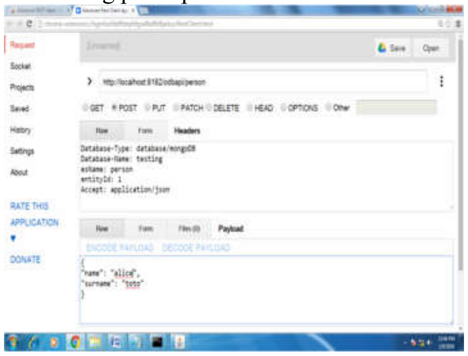


[illegible]



[illegible]

Performing post operation:



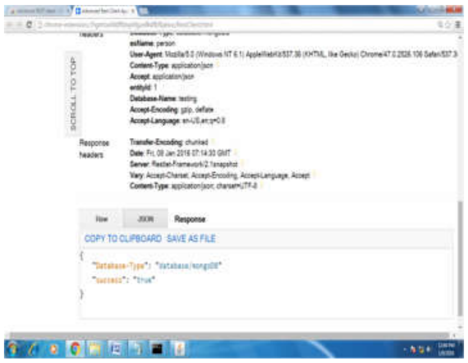
http://localhost:8182/odbapi/person

GET [✓ POST] PUT PATCH DELETE HEAD OPTIONS Other

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

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

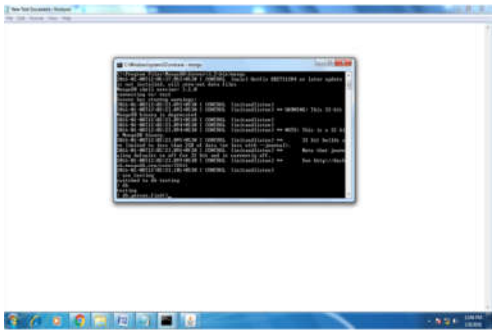
After successfully posting the data:



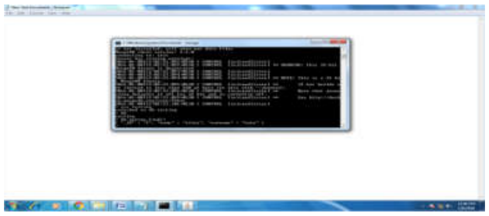
```
esName: person
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.3
Content-Type: application/json
Accept: application/json
entityId: 1
Database-Name: testing
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.8

Response headers
Transfer-Encoding: chunked
Date: Fri, 08 Jan 2016 07:14:30 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"
}
```

Check in the database:

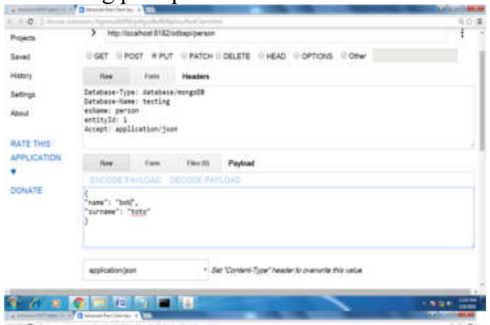


[illegible]



[illegible]

Performing put operation:



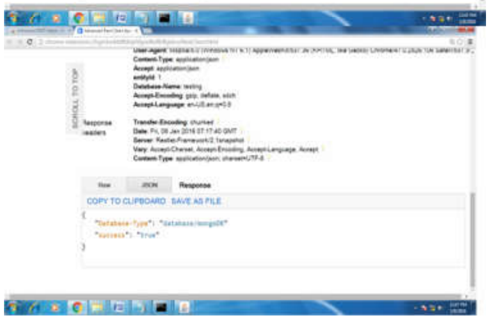
http://localhost:8182/odbapi/person

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"
}
```

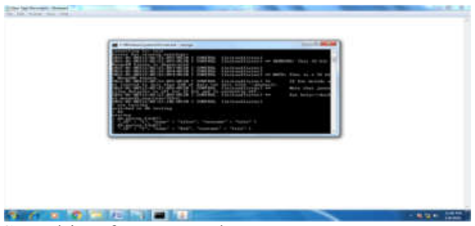
application/json Set "Content-Type" header to overwrite this value.



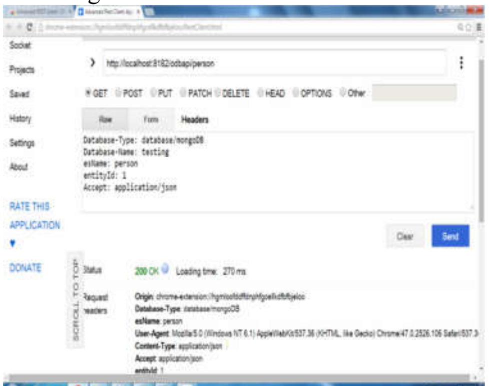
```
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.3
Content-Type: application/json
Accept: application/json
entityId: 1
Database-Name: testing
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8

Response headers
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"
}
```

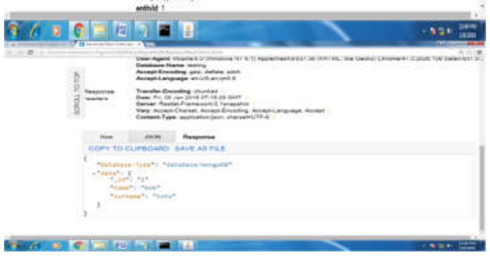
Check in the database after updating:



Searching for a record:

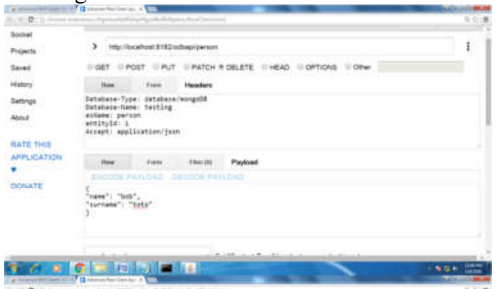


```
http://localhost:8182/odbapi/person

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

Status  200 OK   Loading time: 270 ms

Request headers
Origin: chrome-extension://hgmloofddffdnphfgcellkdfbfbjeloo
Database-Type: database/mongoDB
esName: person
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
```

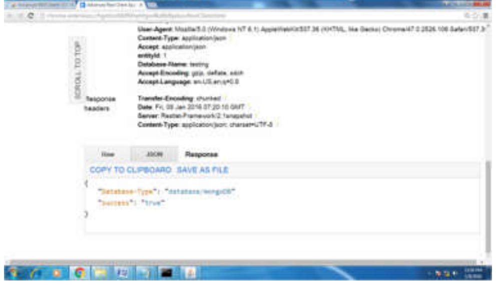


```
Response headers
Transfer-Encoding: chunked
Content-Type: application/json; charset=UTF-8

{
  "Database-Type": "database/mongoDB"
  -"data": {
     "_id": "1"
     "name": "bob"
     "surname": "toto"
  }
}
```

Deleting a record:

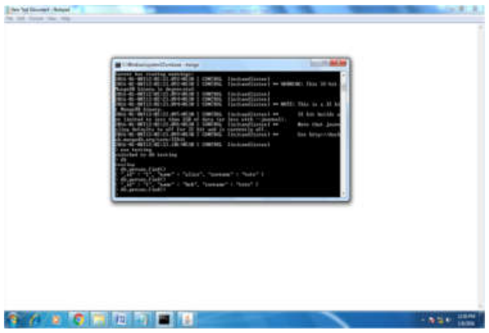


```
http://localhost:8182/odbapi/person

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

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

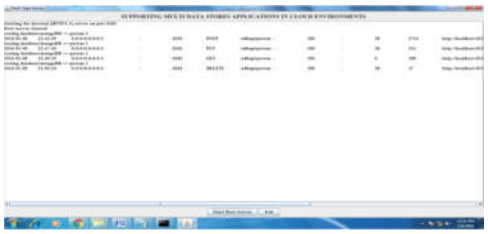


```
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.3
Content-Type: application/json
Accept: application/json
entityId: 1
Database-Name: testing
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8

Response headers
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"
}
```

Check in the database:



Server screen:



CONCLUSION

In this paper, we proposed a generic approach to facilitate the developer task and enable the development of applications using multiple data stores while remaining agnostic to these latter. We introduced three solutions: • ODBAPI for CRUD operations: We defined a generic resources model to represent the different elements of heterogeneous data stores in a Cloud environment. Based on this, we define a unique REST API that enables the management of the described resources in a uniform manner. This API is called ODBAPI and allows the execution of CRUD operations on relational and NoSQL data stores. The highlights of ODBAPI are twofold: (i) decoupling cloud applications from data stores in order to facilitate their development and their migration, and (ii) easing the developer's task by lightening the burden of managing different APIs. It is noteworthy that in the current version of ODBAPI server, we took into account four data stores: MySQL, Riak, CouchDB, and MongoDB.

• Virtual data stores for complex queries execution: We proposed virtual data stores to execute complex queries (including joins) across NoSQL and relational data stores. For this purpose, we defined a unifying data model able to describe the heterogeneous data models of data stores. It is used by the user to express his complex query and by the virtual data store to process it. Once a virtual data store receives a complex query, it constructs an optimal query execution plan, composed by sub-queries at the level of target data sources, conversion and shipping operations and a final query recombining partial results.

• Manifest for data stores discovery and automatic application deployment: Once the developer has completed the development of his application, we provided him the ossibility to express his application requirements in terms of data stores in the *abstract application manifest*. Then, he sends it to the *matching module* that interacts with the *cloud providers discovery module* to elect the appropriate cloud provider to the application requirements. Indeed, the *cloud providers discovery module* discovers the capabilities of data stores of each cloud provider and returns these capabilities in the *offer manifest*. Based on that, the *matching module* implements the *matching algorithm* in order to elect the adequate cloud provider to the application requirements and generates the *deployment manifest* of the application. Once it is done, we deploy the application using the COAPS API that takes as input the *deployment manifest*

REFERENCES

[1] C. Baun, M. Kunze, J. Nimis, and S. Tai, Cloud Computing - Web- Based Dynamic IT Services. Springer, 2011.
[2] A. McAfee and E. Brynjolfsson, "Big data: The management revolution. (cover story)." Harvard Business Review, vol. 90, no. 10, pp. 60–68, 2012.
[3] T. Kraska, M. Hentschel, G. Alonso, and D. Kossmann, "Consistency rationing in the cloud: Pay only when it matters," PVLDB, vol. 2, no. 1, pp. 253–264, 2009.
[4] R. Sellami, S. Bhiri, and B. Defude, "ODBAPI: a unified REST API for relational and NoSQL data stores," in The IEEE 3rd International Congress on Big Data (BigData'14), Anchorage, Alaska, USA, June 27 - July 2, 2014, 2014.
[5] S. Abiteboul and N. Bidoit, "Non first normal form relations: An algebra allowing data restructuring," J. Comput. Syst. Sci., vol. 33, no. 3, pp. 361–393, 1986.
[6] D. Kossmann, "The state of the art in distributed query processing," ACM Comput. Surv., vol. 32, no. 4, pp. 422–469, Dec. 2000.
[7] M. Sellami, S. Yangui, M. Mohamed, and S. Tata, "Paasindependent provisioning and management of applications in the cloud," in 2013 IEEE Sixth International Conference on Cloud Computing, Santa Clara, CA, USA, June 28 - July 3, 2013, 2013, pp. 693–700.
[8] 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.
[9] M. Pollack, O. Gierke, T. Risberg, J. Brisbin, and M. Hunger, Eds., Spring Data. O'Reilly Media, October 2012.