International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-9 Issue-6, April 2020

PaaS Cloud Application and Database Portability: An Initial View

Kiranbir Kaur, Sandeep Sharma, Karanjeet Singh Kahlon

Abstract*: *A new paradigm to cater to the demand for increasingly complex software systems and to shape the way software applications are developed, has emerged, called Cloud Computing. Evolved from the already prevailing and established technologies such as web services, SOA (Service Oriented architecture), virtualization, grid, and cluster computing, Cloud computing proved it to realize the dream of transforming computing as a utility to the customers. Applications developed at Platform as a Service level (Infrastructure as a Service and Software as a Service being the other two levels) face vendor lock-in issue as the proprietary and non-standard APIs offered by providers results in a lack of interoperability and portability among cloud providers. This paper reports on an experiment done to assess the difficulties encountered while porting an application that uses various SQL and NoSQL data stores, message queue service and blob storage of Microsoft Azure, Amazon Web services and Google Cloud platform among each other. The heterogeneity of the incompatible proprietary interfaces makes the porting a non-trivial task. Various problems faced during the portability of the application are discussed and a middleware solution approach to these problems is proposed in this paper.

Keywords* : *Platform as a Service clouds, Application portability, Data portability, Amazon Web Services, Microsoft Azure, Google Cloud Platform

I. INTRODUCTION

A cloud service model represents the particular types of services to be accessed by a user on a cloud platform. The service models of cloud computing are stacked to form a reference model with one layer leveraging the services of the lower layer. The three service models accepted universally are:

Infrastructure as a Service (IaaS): It delivers customizable infrastructure consisting of virtual machines, virtual storage, network devices, load balancers databases, web servers, etc. on demand. Examples of IaaS providers are Amazon Elastic Compute Cloud (EC2), Rightscale, GoGrid, Joyent.

Revised Manuscript Received on April 30, 2020.
***** Correspondence Author
Kiranbir Kaur*, Department of Computer Engg. And Technology, Guru Nanak Dev University, Punjab, Amritsar, 143001, India, E-mail: kiran.dcse@gndu.ac.in.
Dr. Sandeep Sharma, Department of Computer Engg. And Technology, Guru Nanak Dev University, Punjab, Amritsar, 143001, India E-mail: sandeep.cse@gndu.ac.in
Dr. Karanjeet Singh Kahlon, Department of Computer Science And Technology, Guru Nanak Dev University, Punjab, Amritsar, 143001, India E-mail: karankahlon@gndu.ac.in

Platform as a Service (PaaS): It provides an environment where developers are provided with virtual machines, operating systems, application services, development frameworks, etc. to create and deploy applications without worrying about the processors and memory required by their applications. Examples of PaaS providers include Amazon Beanstalk, Microsoft Azure platform, Google App Engine (GAE).

Software as a Service (SaaS): It is the top layer of the stack where applications and the services are provided on demand through Web portals. It has the highest level of abstraction alleviating customers of the burden of software maintenance and customer is concerned with the application data and user interaction. Examples of SaaS applications are GoogleApps, Zoho Office and SalesForce.com.

All these models offer a layer of abstraction to their users for building application systems. Laura DiDio, principal of research and consultancy ITIC observed that

"A decade ago, SaaS was the most popular of the three. But over time, that's changed, and PaaS is now the most popular and the service of choice because it's a "one-stop shop."[1]

But, the PaaS layer suffers from the vendor lock-in problem due to the heterogeneity of proprietary and nonstandard APIs offered by the providers. Standardization and intermediation (Abstraction based approaches and Model-based approaches) [14] are the main application portability approaches that have been proposed in the literature to eliminate the vendor lock-in problem. Ensuring portability across cloud platforms is a panacea to mitigate the vendor lock-in. This would allow the users to switch among the providers when the user' needs change, increasing their confidence towards cloud services. Databases are also an integral part of most of the enterprise applications. The data stores offered by the PaaS providers are limited and proprietary as well. So, the migration of data and database portability also becomes a challenge while porting an application.

The focus of this paper is to assess the challenge of application portability and database migration in the context of Platform as a Service cloud platforms and enlist the problems encountered.

The paper is structured as follows. Section II discusses application portability and database migration challenges in PaaS clouds along with the related work towards these challenges. Section III describes the application and database portability scenarios and the details of the difficulties

[1] https://www.techrepublic.com/article/saas-paas-iaas-the-differences-between-each-and-how-to-pick-the-right-one/

encountered while doing so. It also discusses briefly a proposed middleware solution to these alleviate difficulties. Section IV concludes the paper.

II. THE CHALLENGE OF APPLICATION PORTABILITY AND DATABASE MIGRATION IN PAAS CLOUD PLATFORMS

Developers have a myriad of options available for providers each providing distinct features of similar services like databases, message queues, blob storage, etc. [16] to use. In this kind of scenario, it makes sense to develop an approach that automatically combines resources from many providers - in other words, generate an application that consumes resources from many providers (behind) but maintains its global functionality (the user does not perceive that it is distributed). As an example, imagine that to store images in Azure is cheaper and better than in GAE, but to perform processing and other tasks in GAE is better than in Azure. It makes sense for you to combine two resources provided in a general application. So, you store your things in azure, perform general processing tasks in GAE and, for example, deploy your application core in Heroku. Thus, you have a general application that takes the best advantages from each provider involved in the composition. This is the magic of the idea - this combination can enable a mosaic of resources that can decrease the final costs of the application.

However, due to several reasons [20] like non-standardized service descriptions, non-standardized technologies and heterogeneous APIs results in a lack of interoperability and portability among the cloud providers. Further, there are heterogeneous storage models, data types, remote APIs for data manipulation and query languages [3] which makes data portability an expensive and time-consuming task. This also results in impedance in migrating application components from one cloud provider to another.

There have been several attempts by the researchers to overcome this impedance of vendor lock-in to the widespread adoption of the cloud computing paradigm. These works are discussed in this section.

A. Application Portability.

Application portability can be considered in the context of IaaS and the portability of the PaaS application [24].

In the former case, an application is hosted inside of a provider's virtual machine (VM) and the user has no control over the VM. This application requires substantial efforts to be ported on another provider due to the heterogeneous technologies supported by the providers. There are fewer works on this type of portability than the second case. [26] proposed and conducted several tests on a development method to test the interoperability between the systems migrating on different hypervisors. They also compared their approach with a previous approach TIOSA [27] and claimed that their approach has more benefits than it. *Cloud Application Requirement Language (CARL)* is introduced in [4] which can be used by the cloud application developer to define an application's software and hardware requirements without being specific about IaaS configurations. [2] proposed the OVF (Open Virtualization Format) toolkit as an open tool that can be leveraged for the portability and deployment of applications among disparate virtualization platforms. [15] presented an object-oriented abstraction layer to hide the implementation details of cloud infrastructures and let the developers develop cloud infrastructure agnostic applications. [7] highlighted SDN (Software Defined Networking) enabled networks as an important aspect to realize cross-cloud applications. This work also evaluated the infrastructure support required and how the deployment of the applications be distributed.

The latter case of application portability involves the applications deployed on PaaS providers such as AWS (Amazon Web Services) Beanstalk and Azure App service. To port this application or to switch the components of the application among various providers requires the developer to know the details of each platform and libraries. [11] categorized the problems posed to portability in PaaS platforms. They also implemented a web interface for users to leverage the machine-readable PaaS profiles that the authors clustered into a set. The same authors in [10] proposed a unified interface that helps a user in deciding on the aptest cloud provider to deploy the applications. RESTful APIs and a Ruby wrapper library are used to implement the adapters for the clouds viz. Cloud Foundry, Heroku, CloudControl and OpenShift. [12] attempted to move a legacy application to the cloud by splitting it. Optimization algorithms could be used for splitting the applications. [17] presented a solution approach for hybrid clouds (composing of a private and a public cloud) using a middleware with a uniform API. [8] proposed an abstraction layer which they called PaaSManager by defining a set of fundamental operations and aggregating them to define a common API. [18, 19] leveraged semantics and developed a Domain Specific Language (DSL) called MobiCloud to achieve application portability.

B. Data Portability

Data portability is defined as "Data portability is the ability to move data among different application programs, computing environments or cloud services. In a cloud computing context, this ability is the data portion of cloud portability, which makes it possible for customers to migrate data and applications between or among cloud service providers (CSPs)."[2]

There are numerous of research works done for SQL to NoSQL and vice versa data migration [6, 13, 21, 22] but this paper deals with the data migration among PaaS clouds. [23] proposed design patterns as a solution for data portability among column oriented and graph databases which are categories of NoSQL databases. This work just proposed a solution and did not give much implementation details. [25] also leveraged cloud data patterns to facilitate application data migration. [1] is another significant research towards data portability. The authors proposed a common data model and a standardized API for both kinds of databases viz. SQL and NoSQL. The authors in [5] proposed a NoSQL data migration framework to support different categories of NoSQL databases viz. document oriented, column oriented and graph databases. It also covered the technical, architectural and legal challenges while migrating data to the cloud. [9] presented an abstraction layer called CSAL (Cloud

[2] https://searchcloudcomputing.techtarget.com/definition/data-portability

Abstraction Layer) to facilitate portability of cloud applications. It provides Blob, Table and Queue abstractions among heterogeneous platform providers specifically EC2 and Windows Azure.

III. EXPERIMENTATION WITH PLATFORM SERVICES AND CLOUDS

We developed a toy application in .NET core framework and tried to set it up on clouds. Figure 1 shows the services and database requirements of this application.

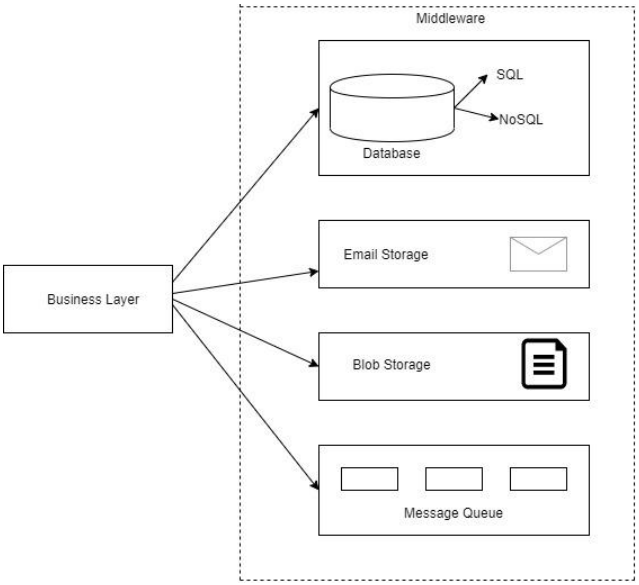

Figure 1. Example of an application leveraging multiple platform services and databases services.

Three clouds are used for the experimentation: Microsoft Azure, Amazon Web Services and Google Cloud Platform.

Both kinds of prevalent database services viz. SQL and NoSQL data stores, other platform services viz. message queue, Email and blob storage service are leveraged covering the following three combination scenarios. Every scenario describes an application with a different combination for each of the chosen cloud (Azure, Amazon and Google):

1. Application using SQL based data store, a test message queue and blob storage service.
2. Application used NoSQL based data store, a test message queue and blob storage service.
3. Application used both SQL and NoSQL based data store, a test message queue and blob storage service.

The SQL PaaS services covered are: Azure SQL Database, Amazon RDS for SQL Server, and Google Cloud SQL for SQL Server.

The NoSQL PaaS services covered are: Azure Cosmos DB (Mongo API), Amazon DocumentDB (Mongo API), MongoDB Atlas on GCP, Azure Cosmos DB (Cassandra API), and Cassandra by Bitnami on Google GCP (Google Cloud Platform).

The Message Queue services covered are: Azure Message Queue, Amazon Simple Queue Service, and Google Pub/Sub

The Blob Storage services covered are: Azure Blob Storage, Amazon S3, and Google Cloud Storage. Figures 2, 3, 4, 5 and 6 show the screenshots for setting up these services for the Azure cloud. Figure 7, 8 and 9 show the screenshots for Google Cloud setup. Similar steps were also followed for the Amazon.

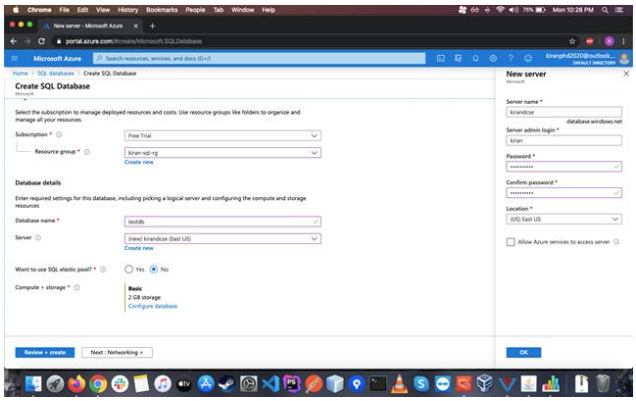

Figure 2. Setting up SQL Database in Azure. Here SQL server and SQL Database are being setup in a single window.

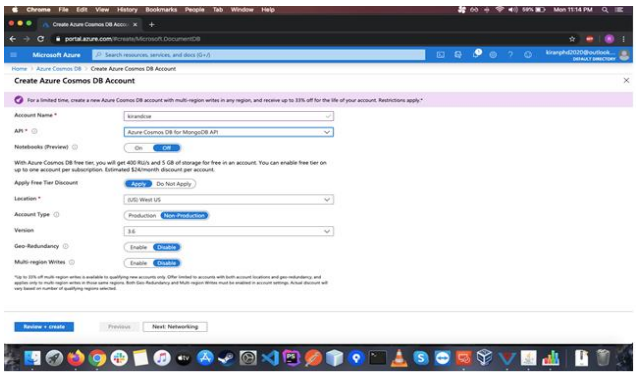

Figure 3. Setting up Azure Cosmos DB with Mongo API in Azure.

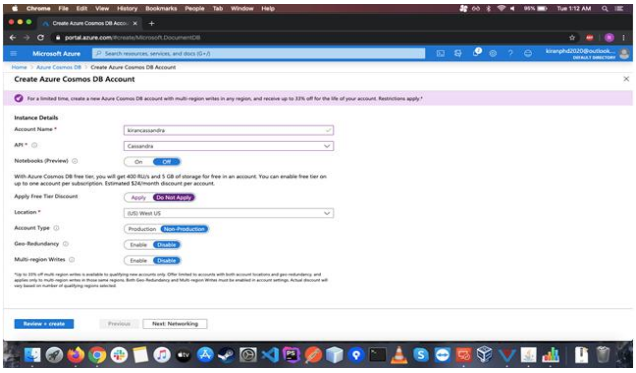

Figure 4. Setting up Azure Cosmos DB with Cassandra API in Azure.

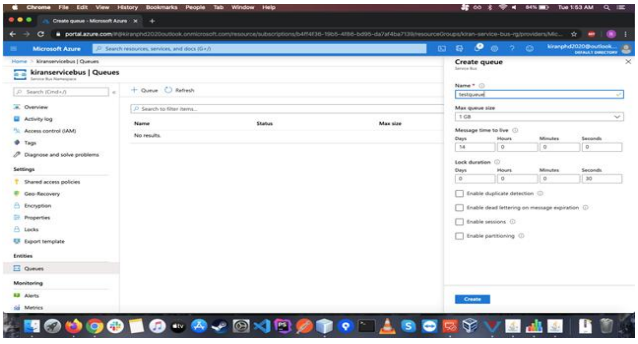

Figure 5. Setting up Service Bus (Message Queue) in

Azure. Here we are setting up Message Queue namespace and Message Queue.

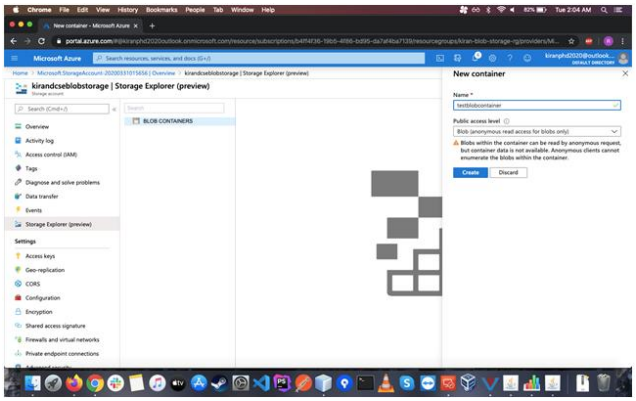

Figure 6. Setting up Blob Storage in Azure. Here we are setting up Storage Account and Blob Container.

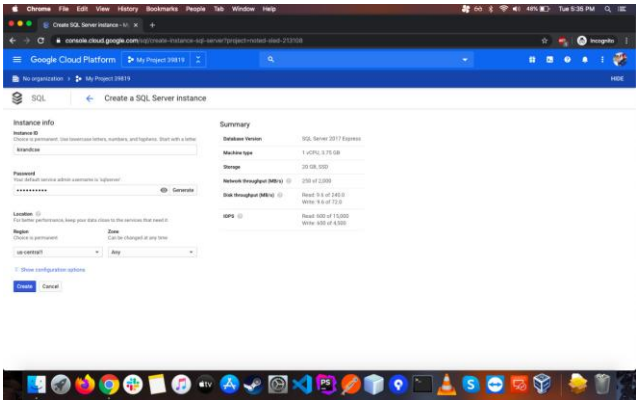

Figure 7. Creating SQL instance in Google Cloud.

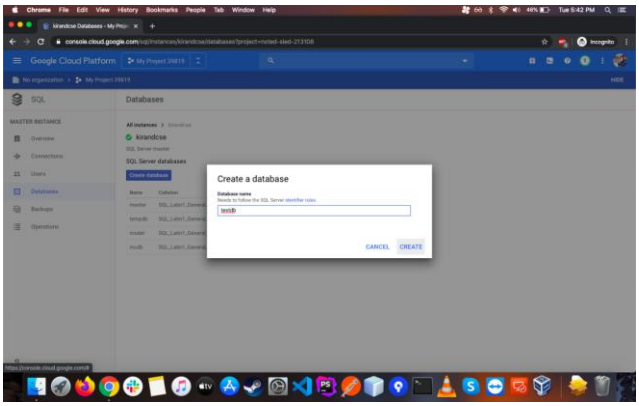

Figure 8. Creating a database in SQL in Google Cloud. It was not available during the creation of SQL server. However in Azure, database and server are configured in same wizard window.

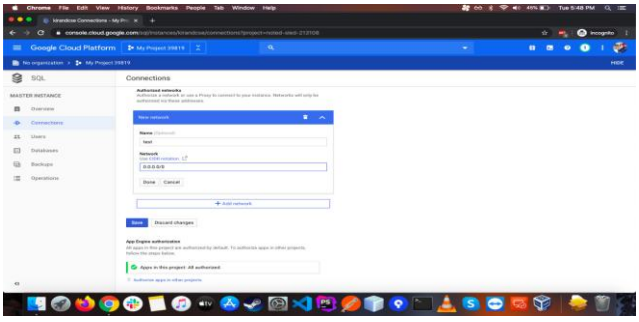

Figure 9. Creating a network settings to allow it connect from outside the cloud. In Azure, it is configured when the server is being created.

A. Scenario 1 Analysis:

Application in this scenario used SQL Server database as data store. The setup of the SQL services on the clouds was not that complex except for the Google Cloud. Azure SQL Database was setup very quickly without any special settings required. Amazon RDS required some additional settings like snapshot settings, VPC settings, Public Access settings etc. Google Cloud SQL setup was the most complex as it required backup settings, Public Access settings, Network IP Range settings, etc.

The application was hosted on Azure first. It was then migrated to AWS and lastly to Google Cloud. Data store migration required changes in connection string but data migration required to import the backup from source SQL data store and export the backup on target SQL data store.

Message Queue, and Blob Storage service required a complete change in the code as every cloud has different settings for the message queue service, and blob storage service. Also, the implementation and work process is different for each mentioned service. So the migration for Message Queue service, and Blob Storage service required drastic changes in the application to make it compatible for each cloud.

B. Scenario 2 Analysis:

Application in this scenario used Mongo based NoSQL database as data store. The setup of the PaaS services on the clouds was rather easier than SQL PaaS services. It required minimal settings to setup.

The application was hosted on each cloud one after another. Data store migration required changes in the connection string as well as in source code of the application. Data migration also required a separate tool to migrate data from source Mongo data store to destination data store.

Message Queue, and Blob Storage services scenario was same as was in Scenario 1.

C. Scenario 3 Analysis:

In this scenario, the application used both SQL and Mongo based NoSQL as data stores. The setup of the data store services here is same as in Scenario 1 and 2.

The application was hosted on each cloud one by one. Data store and data migration between data stores was not supported and required additional code to be implemented in the application.

Message Queue, and Blob Storage services scenario was same as Scenario 1.

D. Problems faced during experimentation

There were several problems faced in this experiment. First one is the data migration between data stores. Second was the migration of Message Queue, and Blob Storage among different clouds. Service availability was a major problem that was faced in this experiment. For example, Cassandra PaaS was not available in Amazon Web Services. However, it is provided in EC2 instance but that becomes an IaaS. Database Portability was another issue faced during the experimentation. Data migration in SQL server was relatively easy as backup file can be generated from the source database. But in Mongo based NoSQL data, an additional tool is required to migrate data between different Mongo based PaaS.

Data migration between SQL and NoSQL also required additional code to be implemented in the application. No official service by providers is available for heterogeneous data migration.

Application Portability was the biggest issue as migration between message queue, and blob storage PaaS. It required a complete rework of the service implementation in the source code of the application to achieve the application service portability.

E. Proposed Solution

We propose a solution to tackle all of the above mentioned problems by developing a middleware that abstracts the implementation of database and application PaaS services and provide a single interface for the user to use the PaaS service in his/her application (Figure 10).

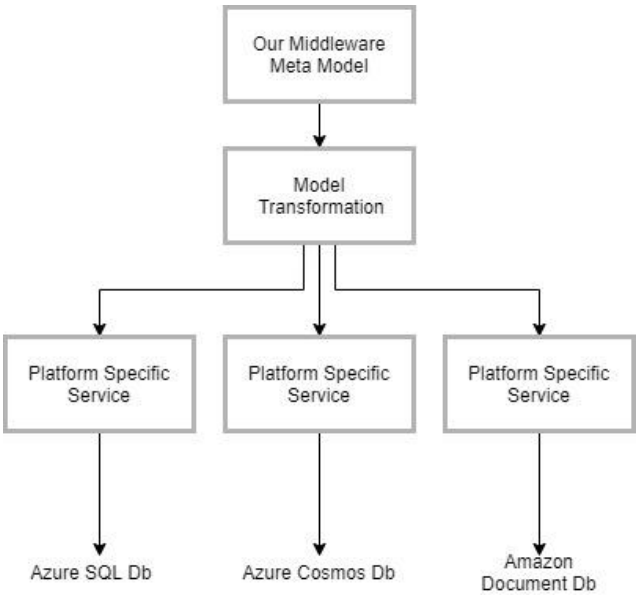

Figure 10. Middleware approach for developing PaaS clouds portable applications.

.

Middleware approach is chosen to tackle this problem because it is easy to distribute it via repositories and easy to update by the developers. If any change is done by PaaS provider, it can be updated in the middleware and then distributed to the middleware users by repositories.

The middleware will consist of a data model for each data database PaaS and will be exposed to the user via an interface. The user will need to update the configuration file to modify the settings of data stores, message queue, and blob storage service.

The proposed middleware will handle all the CRUD operations along with Joins operations. It will help users to focus on the business logic rather than PaaS implementation.

IV. CONCLUSION

Application and database portability among PaaS providers is a difficult problem that is necessary to be tackled as the cloud computing paradigm is soaring high. In this paper, we discussed the challenge of porting an application along with its databases to the prominent PaaS providers (Amazon Beanstalk, Google Cloud Platform and Microsoft Azure). The high degree of heterogeneity among these providers make it non trivial. We presented some related work towards application and database portability. Then we reported on an ongoing experiment to carry out various portability scenarios to analyze the difficulties encountered. Finally we proposed a middleware approach as a solution to alleviate these difficulties. This middleware would enable developers to develop cloud agnostic applications by offering the developers with meta model classes and abstracted services. As future steps, we will develop this proposed middleware to realize the portability among clouds. A data migration tool would also be developed for migrating the existing data in the previous cloud database while moving to a new database in the same cloud or a different one.

REFERENCES

1. Ebtesam Alomari, Ahmed Barnawi, and Sherif Sakr. 2015. CDPort: A Portability Framework for NoSQL Datastores. Arab. J. Sci. Eng. 40, 9 (2015), 2531–2553. DOI:https://doi.org/10.1007/s13369-015-1703-0
2. Gaetano F. Anastasi, Emanuele Carlini, Massimo Coppola, Patrizio Dazzi, and Marco Distefano. 2014. An OVF Toolkit Supporting Inter-Cloud Application Splitting. In IEEE 3rd International Conference on Cloud Networking(CloudNet), 96–101.
3. Darko Androcec. 2013. Data Portability among Providers of Platform as a Service. Fac. Mater. Sci. Technol. (2013), 7–11.
4. Demetris Antoniades, Nicholas Loulloudes, Athanasios Foudoulis, Chrystalla Sophokleous, Demetris Trihinas, George Pallis, and Marios Dikaiakos. 2015. Enabling Cloud Application Portability. In IEEE/ACM 8th Interntional Conference on Utility and Cloud Computing, 354–360.
5. Aryan Bansel, Horacio Gonzalez-Velez, and Adriana E. Chis. 2016. Cloud-Based NoSQL Data Migration. Proc. - 24th Euromicro Int. Conf. Parallel, Distrib. Network-Based Process. PDP 2016 (2016), 224–231. DOI:https://doi.org/10.1109/PDP.2016.111
6. Wu Chun Chung, Hung Pin Lin, Shih Chang Chen, Mon Fong Jiang, and Yeh Ching Chung. 2014. JackHare: a framework for SQL to NoSQL translation using MapReduce. Autom. Softw. Eng. 21, 4 (2014), 489–508. DOI:https://doi.org/10.1007/s10515-013-0135-x
7. Felix Cuadrado, Alvaro Navas, Juan C.Duenas, and Luis M.Vaquero. 2014. Research Challenges for Cross-Cloud Applications. In IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 19–24.
8. David Cunha, Pedro Neves, and Pedro Sousa. 2014. PaaS manager: A platform-as-a-service aggregation framework. Comput. Sci. Inf. Syst. 11, 4 (2014), 1209–1228. DOI:https://doi.org/10.2298/CSIS130828028C
9. Zach Hill and Marty Humphrey. 2010. CSAL: A cloud storage abstraction layer to enable portable cloud applications. Proc. - 2nd IEEE Int. Conf. Cloud Comput. Technol. Sci. CloudCom 2010 (2010), 504–511. DOI:https://doi.org/10.1109/CloudCom.2010.88
10. Stefan Kolb and Cedric Rock. 2016. Unified Cloud Application MAnagement. In 2016 IEEE World Congress on Service Computing, 1–8.
11. Stefan Kolb and Guido Wirtz. 2014. Towards application portability in platform as a service. Proc. - IEEE 8th Int. Symp. Serv. Oriented Syst. Eng. SOSE 2014 (2014), 218–229. DOI:https://doi.org/10.1109/SOSE.2014.26
12. Frank Leymann, Christoph Fehling, Ralph Mietzner, Alexader Nowak, and Schahram Dustdar. 2011. Moving Applications to the Cloud: An Approach based on Application Model Enrichment. Int. J. Coop. Inf. Syst. 3, 2 (2011), 307–356.
13. Muhammad Mughees. 2013. DATA MIGRATION FROM STANDARD SQL TO NoSQL. Retrieved from http://ir.obihiro.ac.jp/dspace/handle/10322/3933
14. Riccardo Munisso and Adriana E. Chis. 2017. CloudMapper: A Model-Based Framework for Portability of Cloud Applications Consuming PaaS Services. Proc. - 2017 25th Euromicro Int. Conf. Parallel, Distrib. Network-Based Process. PDP 2017 (2017), 132–139. DOI:https://doi.org/10.1109/PDP.2017.94
15. Binh Minh Nguyen, Viet Tran, and Ladislav Hluchy. 2013. Development and deployment of cloud services via abstraction layer. In International Conference on Computing, Management and Telecommunications (ComMan Tel), 246–251.

16. Elias Nogueira, Ana Moreira, Daniel Lucrédio, Vinícius Garcia, and Renata Fortes. 2016. Issues on developing interoperable cloud applications: definitions, concepts, approaches, requirements, characteristics and evaluation models. J. Softw. Eng. Res. Dev. 4, 1 (2016). DOI:https://doi.org/10.1186/s40411-016-0033-6
17. Ansar Rafique, Stefan Walraven, Bert Lagaisse, Tom Desair, and Wouter Joosen. 2014. Towards portability and interoperability support in middleware for hybrid clouds. Proc. - IEEE INFOCOM (2014), 7–12. DOI:https://doi.org/10.1109/INFCOMW.2014.6849160
18. Ajith Ranabahu, E. Michael Maximilien, Amit Sheth, and Krishnaprasad Thirunarayan. 2015. Application portability in cloud computing: An abstraction-driven perspective. IEEE Trans. Serv. Comput. 8, 6 (2015), 945–957. DOI:https://doi.org/10.1109/TSC.2013.25
19. Ajith Ranabahu and Amit Sheth. 2010. Semantics centric solutions for application and data portability in cloud computing. Proc. - 2nd IEEE Int. Conf. Cloud Comput. Technol. Sci. CloudCom 2010 April (2010), 234–241. DOI:https://doi.org/10.1109/CloudCom.2010.48
20. Rajiv Ranjan. 2014. The Cloud Interoperability Challenge. IEEE Cloud Comput. 1, 2 (2014), 20–24. DOI:https://doi.org/10.1109/MCC.2014.41
21. Leonardo Rocha, Fernando Vale, Elder Cirilo, Dárlinton Barbosa, and Fernando Mourão. 2015. A framework for migrating relational datasets to NoSQL. Procedia Comput. Sci. 51, 1 (2015), 2593–2602. DOI:https://doi.org/10.1016/j.procs.2015.05.367
22. John Roijackers and H.L.George Fletcher. 2013. XML Query Processing: On Bridging Relational and Document-Centric Data Stores. In LNCS 7968 - Big Data. 135–148.
23. Mahdi Negahi Shirazi, Ho Chin Kuan, and Hossein Dolatabadi. 2012. Design patterns to enable data portability between clouds' databases. Proc. - 12th Int. Conf. Comput. Sci. Its Appl. ICCSA 2012 (2012), 117–120. DOI:https://doi.org/10.1109/ICCSA.2012.29
24. Elias Adriano Nogueira Da Silva, Victor Gomes Da Silva, Daniel Lucrédio, and Renata Pontin De Mattos Fortes. 2013. Towards a model-driven approach for promoting cloud PaaS portability. Proc. 2013 39th Lat. Am. Comput. Conf. CLEI 2013 October 2013 (2013). DOI:https://doi.org/10.1109/CLEI.2013.6670667
25. Steve Strauch, Vasilios Andrikopoulos, Thomas Bachmann, and Frank Leymann. 2013. Migrating Application Data to the Cloud Using Cloud Data Patterns This publication and contributions have been presented at CLOSER 2013 Migrating Application Data to the Cloud using Cloud Data Patterns. (2013).
26. Soffa Zahara, Istas Pratoma, and Djoko suprajitno Rahardjo. 2015. Application and Data Level Interoperability on Virtual Machine in Cloud Computing Environment. In 1st International Conference on Wireless and Telematics(ICWT), 1–5.
27. Alexander Lenk, Gregory Katsaros, Michael Menzel, Jannis Rake, Ryan Skipp, Enrique Castro-Leon, P.GopanV. 2014. TIOSA: Testing VM Interoperability at an OS and Application Level -- A Hypervisor Testing Method and Interoperability Survey. *IEEE International conference on Cloud Engineering. 245-252*

AUTHORS PROFILE

Kiranbir Kaur is currently working as an assistant professor in the department of Computer Engg. And Technology, Guru Nanak Dev University Amritsar. She has completed M.Tech (CSE) and pursuing Ph.d. She has more than 10 years of experience in teaching. Her area of interest is Cloud Computing and databases.

Dr. Sandeep Sharma is currently working as a professor and Head of the department of Computer Engg. And Technology, Guru Nanak Dev University Amritsar. He has more than 15 years of experience in teaching and industry. His area of interest are Cloud Computing, Parallel Processing and Wireless Sensor Network.

Dr. Karanjeet Singh Kahlon is currently working as a professor in the department of Computer Science and Applications, Guru Nanak Dev University Amritsar. He has more than 20 years of experience in teaching. His area of interest are Networking, Distributed Computing