Survey on Middleware Systems in Cloud Computing Integration

Jonathan A.P. Marpaung*, Mangal Sain*, Hoon-Jae Lee*

*Cryptography & Network Security Lab, Dongseo University

San 69-1, Jurye 2-dong, Sasang-gu, Busan 617-716, Korea

jonathan@spentera.com, mangalsain1@gmail.com, hjlee@dongseo.ac.kr

***Abstract*— Cloud computing is an increasingly attractive solution to providing cheap, scalable, and easy to deploy computing services. This paper presents a survey on the middleware technology used to integrate different cloud platforms/services, applications running in the cloud, and the cloud with on-premise data centers. We discuss the ongoing research projects and solutions being developed such as Altocumulus, AppScale, Cloudify, and mOSAIC. We analyze the features already available in current solutions as well as the future technology and standards needed to ensure seamless integration and more widespread adoption of cloud computing solutions.**

***Keywords*— Cloud computing, Middleware, Integration**

I. INTRODUCTION

Cloud computing is a computing model that refers to applications offered over the internet as well as the hardware and systems software in the data centers providing these services [1]. Cloud computing generally falls under three classes: Software as a Service (SaaS), Infrastructure as a Service (IaaS), and Platform as a Service (PaaS). Examples of SaaS include applications such as Google Apps, Salesforce, and Microsoft Office 365. Examples of IaaS include Amazon CloudFormation, Google Compute Engine, and Rackspace Cloud. Several popular PaaS are Google App Engine, Microsoft Azure, and Amazon Elastic Beanstalk.

Middleware was originally described as a layer of software that sits between platforms and applications in distributed systems [2]. Nowadays that definition has broadened to encompass software that provides abstraction over heterogeneous software and hardware systems with a unifying interface.

In the context of cloud computing, we limit our discussion on cloud middleware to middleware that provides integration between different cloud application components, services/providers as well as between clouds and on-premise systems in private company data centers. Traditional virtualization is not within the scope of this discussion. This model is illustrated in Figure 1.

Cloud computing is increasingly becoming a commonplace component of the computing services used in today's organizations. It is often seen as a cheaper, easier, and more scalable method of deploying applications and services. However, despite its apparent attractiveness, challenges with security, integration, and data migration are some of the main reasons behind unexpected costs and the source of reluctance of many companies in adopting this computing model [3]. In this paper we focus the discussion on the role of middleware in addressing the latter two challenges

Integration is a major challenge when organizations have some applications hosted on their own infrastructure, some pushed onto the cloud, and some provided by different cloud computing services. This is particularly more visible when attempting to integrate SaaS across different providers. Data or code lock-in occurs when customers are limited to specific platforms and cannot migrate data or applications to other providers or to in-house data centers. The Industry has begun to recognize the importance of collaboration, standards, and avoiding lock-in, with many companies supporting the Open Cloud Manifesto.

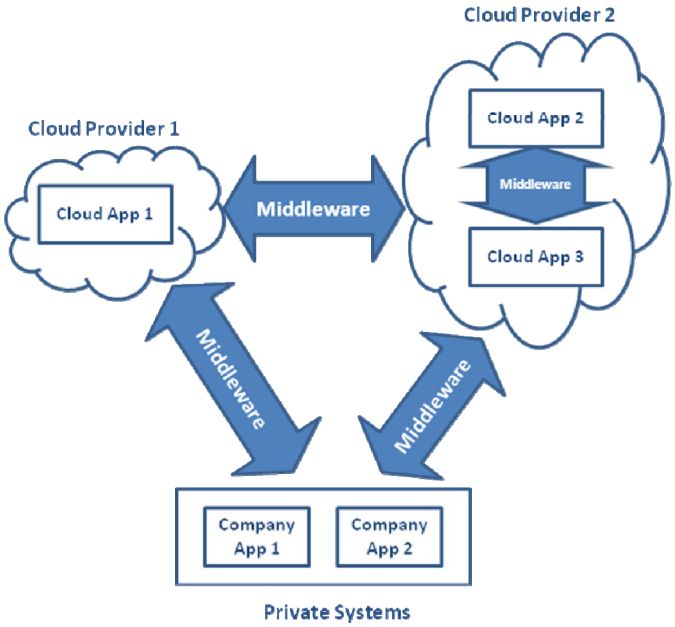

Figure 1. Middleware services in a cloud environment

Moving forward, the cloud computing industry needs to offer better integration features in order to achieve widespread adoption by organizations currently not using or only partially using the cloud. Therefore the state and development of

middleware systems will play an important role in determining the future of cloud computing.

In this paper we survey the current cloud middleware systems developed such as Altocumulus, AppScale, GigaSpaces, and mOSAIC. We analyze the characteristics and techniques found in these implementations. Furthermore we discuss the future direction cloud middleware development needs to take in order to ensure seamless integration among the various components of cloud computing.

II. Cloud Integrating Middleware Systems

Several middleware systems for cloud integration have been developed. In this section we discuss the features of these systems.

A. IBM Altocumulus

The Altocumulus middleware provides users with the ability to deploy applications (primarily web applications) to a variety of clouds [4].

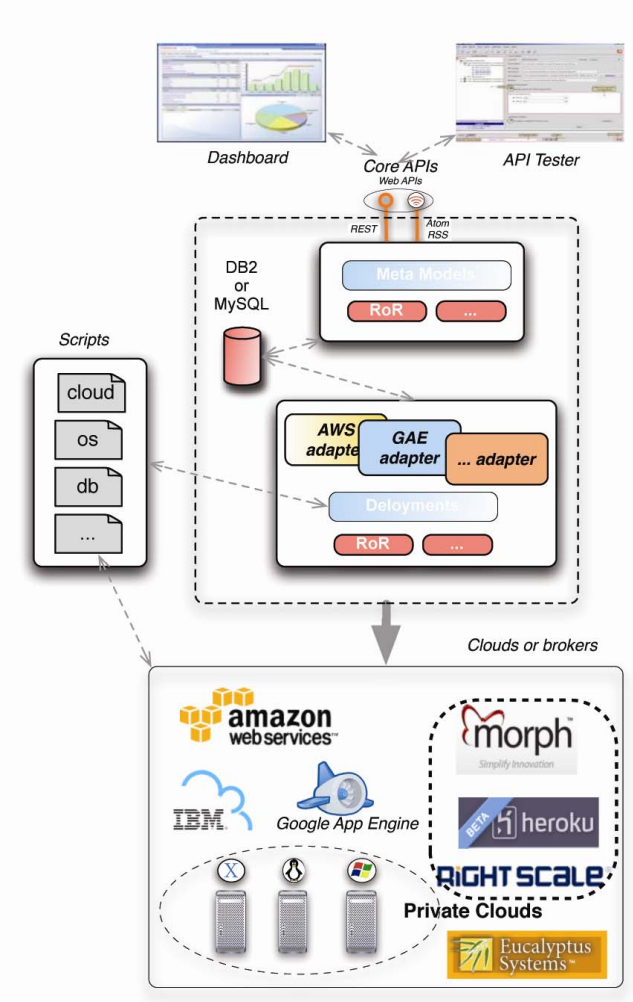

Figure 2. Altocumulus Architecture

Altocumulus has the goal of homogenizing the clouds with a layer and API, thus allowing higher level application development. Users would then have the flexibility to migrate application deployments across different clouds platforms with the middleware hiding the complexity of this process. The cloud platforms currently supported are Amazon EC2, Eucalyptus, Google AppEngine, and IBM HiPODS.

The middleware consists of three major components: Core, Dashboard, and Core-API. The Core is the layer that interacts with different cloud providers which also contains a private RESTful API. The Dashboard provides a web based user interface and uses the Core via the private API. The Core-API is a well documented, public API for the core. Core-API allows for control over middleware functionalities for external applications. Manually programmed adapters are used to interface between different clouds. A support component, called the image repository, manages and stores the images for various clouds. The Altocumulus researchers depict how these components interact with each other in Figure 2.

The project does not yet support hybrid application deployments across different clouds e.g. one application running concurrently on 3 cloud platforms. Another barrier this middleware does not address is lock-in due to cloud providers restricting application development to limited libraries.

B. AppScale

AppScale is an open-source hybrid cloud platform developed by the RACELab of UC Santa Barbara. AppScale implements the Google App Engine (GAE) open APIs and provides an infrastructure and toolset for running GAE applications across "non-Google clouds" [5]. Although essentially a PaaS framework, AppScale contains integrating middleware features that allow distributed deployment of GAE applications on different IaaS clouds such as Amazon's EC2 and EUCALYPTUS.

AppScale consists of three main components, the AppServer (ASs), the database management system, and the AppLoadBalancer (ALB). The ASs executes the GAE applications and communicates with a Database Master over HTTPS for accessing and storing data. The ALB initiates connections to GAE applications running in ASs. In addition to these components AppScale has an AppController (AC) for inter-component communication and the AppScale Tools, which are used by developers to deploy AppScale and manage GAE applications. The interaction between these components is illustrated in Figure 3.

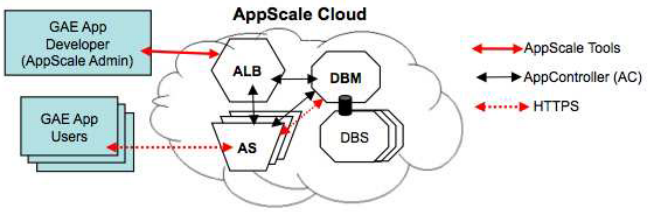

Figure 3. AppScale Architecture

The main integrating component in AppScale is the AppServer, which is an extension of the free development server distributed with the Google AppEngine SDK. The

extensions developed by the RACELab enable execution of GAE applications on any virtualized cluster. In addition, a generic datastore interface is provided that the developers claim to support any database technology. The interface has been implemented to HBase and Hypertable. Plugins are also available for MySQL, Cassandra, and Voldemort.

AppScale offers a hybrid cloud platform environment for GAE applications. This is a good model for modular design with its middleware components such as the AppController and datastore interface.

C. GigaSpaces Cloudify

Cloudify is described as 'The Open PaaS Stack' by GigaSpaces. It is a platform that is designed to support any application regardless of the application stack (i.e. languages and dependencies), can be deployed on any IaaS cloud, and provides full control over the underlying infrastructure to its users [6]. This allows existing applications running on private servers to be directly migrated to the cloud with no modification to code. Vendor lock-in can be avoided due to this flexibility and high level of customizability.

Application 'recipes' are used to specify to Cloudify the details required to run the application including middleware services, dependencies, how to monitor the services, as well as other details. Recipes are described in a Groovy based DSL (domain specific language) which can be customized and extended. Essentially infrastructure is treated like code with configuration and installation completely automated after the recipe is handled by Cloudify. Pre-configured recipes are also available for popular middleware components such as JBoss, Tomcat, MySQL, MongoDB, and so on.

The Cloudify Architecture consists of 3 major components: the Universal Service Adaptor, Cloud Controller, and Cloud Driver. A Universal Service Adaptor runs on each node and translates the recipe into actions such as service installation, initialization, and monitoring. It is said to be 'service agnostic'; does not differentiate between the services managed as long as the recipe correctly configures it. This middleware component leaves developers free to focus on coding the application, as the environment does not need to be installed manually.

The Cloud Controller is the central engine that manages the deployment of the application, by continuous monitoring and triggering alerts and scaling rules based on the current load and other parameters. The Cloud Driver is a layer of abstraction that isolates the application from IaaS environments thus providing a foundation for integration with different cloud providers. This design allows for application deployments across multiple cloud environments while keeping the complexity of this process hidden from the higher layers. Figure 4 is an illustration of this architecture presented by GigaSpaces.

Cloudify is already integrated with Microsoft Windows Azure and Amazon EC2. Support for Citrix CloudStack, OpenStack, Rackspace, GoGrid, and Citrix XenServer is being developed.

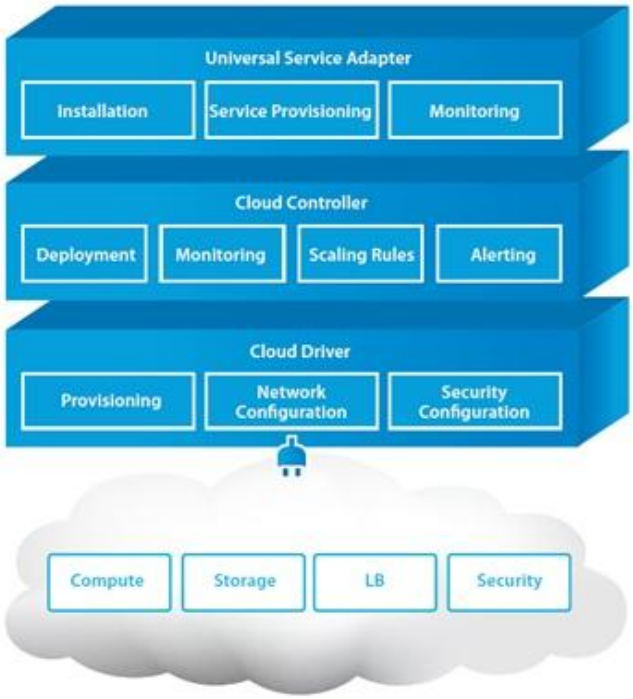

Figure 4. Cloudify Architecture

D. mOSAIC

mOSAIC is an open-source API and platform for designing and developing multi-Cloud-oriented applications [7]. At the core of its features is a language and platform agnostic API. This API is illustrated by the researchers in Figure 5. The main middleware components providing integration features are the Cloudlet, Connector, Interoperability, and Driver API.

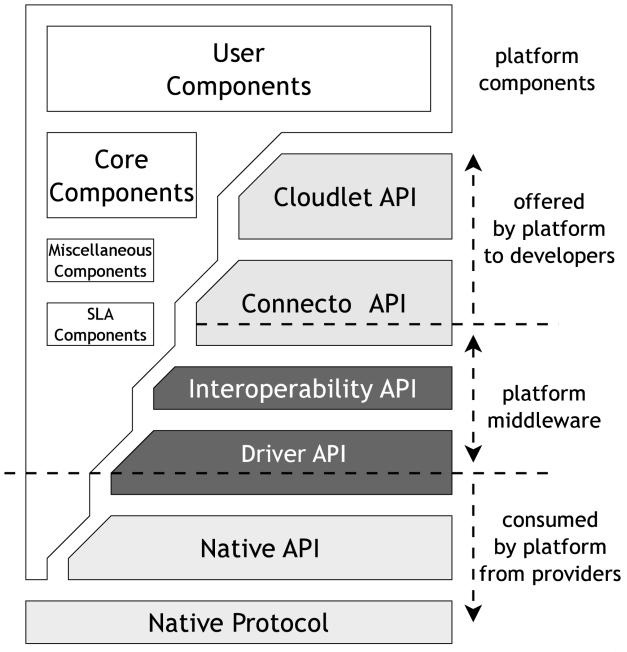

Figure 5. mOSAIC API Layers

The Cloudlet and Connector API layers facilitate the integration into the target language environment which is used by the developers in their applications. The Driver API layer provides abstraction over resource allocation on top of the native resource API. The Interoperability API is the middleware layer that integrates the connector API and compatible driver API implementations that could be written in different languages. It is a remote API that follows the model of RPC with functionalities including marshalling, request/response correlation, and error detection.

Apart from its cloud integration features, mOSAIC framework is promised to have a semantic-oriented ontology for describing cloud resources.

III. CHARACTERISTICS

From the discussion in the previous section, we summarize the characteristics of middleware systems in cloud integration.

A. Support open platform stacks

Several of the aforementioned middleware systems are solutions to the problem of cloud vendor application library lock-in. Cloud middleware is enabling developers to focus on developing the functionality of their applications instead of being concerned with the compatibility of their code and the underlying environment. To this end, cloud PaaS providers would have to provide more application stack options to users or employ a middleware service to allow this level of flexibility. The latter course of action is less costly and more maintainable.

B. Generic interfaces between components

A kind of plug-and-play capability integrating various components in a cloud system is made possible by implementing generic interfaces between components. In the case of AppScale, a generic datastore interface allows the application to use any database technology. Loosely connected APIs and implementations create opportunities for diverse interoperability between components. This allows for collaboration on the cloud between different services that would not normally work together in a native deployment environment.

C. Support infrastructure abstraction

Infrastructure abstraction isolates the application from the cloud thus enabling hybrid cloud deployments. The integrating cloud middleware in this area serves to allow applications to run on private and public clouds as well as across public clouds. Users would have the ability to take the advantages of certain cloud provider features as well as the ability to scale dynamically depending on needs and availability of resources on each cloud.

D. Utilize standard communication frameworks

Middleware APIs in the example systems discussed have mostly implemented mature forms of web services (such as SOAP or REST) or remote calls (such as Sun RPC or AMF). Basing middleware communications on platform independent data structures and protocols is necessary to maintain interoperability regardless of changes to other layers in the cloud.

IV. CONCLUSIONS

The future of cloud computing services is heading towards offerings of flexible development and deployment environments. Moving forward, the specifics of the application stack and of which cloud service to choose from will no longer be a deciding factor. Moreover models such as Sky Computing [8] will be made more feasible. Cloud middleware system integrators will determine the future adoption of cloud computing. These systems should support open platform stacks, provide generic interfaces between components, support infrastructure abstraction, and utilize standard communication frameworks.

ACKNOWLEDGMENTS

This research was supported by a research program of Dongseo University's Ubiquitous Appliance Regional Innovation Center supported by the grants from Ministry of Knowledge Economy of the Korean government and Busan Metropolitan City (No. B0008352).

And it also supported by NRF2012 project (grant number: 2012-0008447)

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," University of California, Berkeley, Tech. Rep. UCB-EECS-2009-28, Feb 2009.

[2] P.A. Bernstein, "Middleware: A Model for Distributed Services," *Comm. ACM,* vol. 39, no. 2, Feb. 1996, pp. 86-97.

[3] L. Herbert, C.F. Ross, M. Grannan, "SaaS Adoption 2010: Buyers See More Options But Must Balance TCO, Security, And Integration," Forrester Research, 2010.

[4] A. Ranabahu, M. Maximilien, A best practice model for cloud middleware systems, in: Proceedings of the Best Practices in Cloud Computing: Designing for the Cloud, ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA, Orlando, FL, USA, 2009

[5] N. Chohan, C. Bunch, S. Pang, C. Krintz, N. Mostafa, S. Soman, and R. Wolski. AppScale: Scalable and Open AppEngine Application Development and Deployment. In International Conference on Cloud Computing, Oct. 2009.

[6] (2012) The Cloudify website. [Online]. Available: http://www.gigaspaces.com/cloudify-open-paas-stack

[7] D. Petcu, C. Craciun, N. Neagul, M. Rak, and I. Lazcanotegui, "Building an interoperability api for sky computing," in Proceedings of the 2011 International Conference on High Performance Computing and Simulation Workshop on Cloud Computing Interoperability and Services, 2011, pp. 405-412.

[8] K. Keahey, M. Tsugawa, A. Matsunaga and J. Fortes, "Sky Computing," IEEE Internet Computing 13, pp. 43-51, September 2009,