COMPARISON OF OPEN-SOURCE PAAS ARCHITECTURAL COMPONENTS

Mohan Krishna Varma Nandimandalam[1] and Eunmi Choi[2]

[1]Graduate School of Business IT, Kookmin University, Seoul, South Korea
nmohankv@kookmin.ac.kr
[2]Corresponding Author, School of Business IT,
Kookmin University, Seoul, Korea
emchoi@kookmin.ac.kr

ABSTRACT

Cloud computing is a widely used technology with three basic service models such as Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). This paper focuses on the PaaS model. Open source PaaS model provides choice of cloud, developer framework and application service. In this paper detailed study of four open PaaS packages such as AppScale, Cloud Foundry, Cloudify, and OpenShift are explained with the considerable architectural component aspects. We also explained some other PaaS packages like Stratos, Stakato and mOSAIC briefly. In this paper we present the comparative study of major open PaaS packages.

KEYWORDS

Cloud Computing, AppScale, Cloud Foundry, Cloudify, OpenShift, Stackato & Stratos

1. INTRODUCTION

Cloud computing is an emerging paradigm in which computers are networked to provide storage and compute services using virtualization technology. Cloud computing must satisfy five essential characteristics. They are on demand service, access network, resource pooling, elasticity and measured services. To achieve these five essential characteristics, cloud computing provides three kinds of service models: Software as a Service (SaaS), Platform as a Service (PaaS) [7] and Infrastructure as a Service (IaaS) [8]. Cloud computing service models are shown in Figure 1. CRM applications are widely used services in the SaaS. Application platform delivered as a service is described as PaaS and it is used to deploy the user code. AppScale [2], Cloud Foundry, Cloudify and OpenShift open-source environments can be used as PaaS. IaaS is used to build their private infrastructure, which reduces the setup cost. IaaS can provide virtualized resources such as computation, storage and communication. Eucalyptus [1], open stack and cloud stack open-sources can be used to provide IaaS.

David C. Wyld et al. (Eds) : CCSEA, CLOUD, DKMP, SEA, SIPRO - 2016
pp. 53–62, 2016. © CS & IT-CSCP 2016 DOI : 10.5121/csit.2016.60206

Figure 1. Cloud computing service models

This paper will focus on the PaaS service model. It is easy to deploy, run and scale application using PaaS. Some of the PaaS have limited language and framework support. They do not deliver key application services needed for cloud applications. They sometime restrict deployment to a single cloud. Whereas open PaaS provides choice of cloud like private, public or hybrid, choice of developer framework like spring, ruby, or java and application services like mongoDB, MySQL, or PostgreSQL for running our applications. This paper deals with the architectural components of major open PaaS packages like AppScale, Cloud Foundry, Cloudify and OpenShift.

The paper is organized as follows. Section 2 introduce AppScale and its components, Cloud Foundry architecture and component explanation given in Section 3, Cloudify open PaaS is explained in Section 4, Section 5 deals with OpenShift, other open PaaS technologies are introduced in Section 6, comparison of open-source PaaS technologies are given in Section 7 and finally Section 8 concludes the paper.

2. APPSCALE

AppScale [3] is a scalable, distributed, and fault-tolerant cloud runtime system that executes over cluster resources. It can be deployed on Xen [4], Kernel-based Virtual Machine (KVM), Amazon EC2 or Eucalyptus. AppScale initial design utilizes the standard three-tier web deployment model in the design. In the later design cycles more components are added to the AppScale. Table 1 shows the AppScale components, language used to design the component and their functionality.

Table 1. AppScale Components

Component	Language	Functionality
AppController	Ruby	Executes on every node and starts automatically when the guest virtual machine boots
AppLoadBalancer	Ruby on Rails	Processes arriving requests from users and forwards them to the application server
AppServer	Python	Running through a number of distant hosts to support automated execution of applications
Database Master	Python	Offers persistent storage for applications, processes protocol buffers from apps and makes requests on its behalf to read and write data to the data store
Database Slave	Python	Facilitate distributed, scalable, and fault tolerant data management
AppScale Tools	Ruby	Starts an AppScale system, deploys and tear down applications, queries the state and performance of AppScale deployment or application, and manipulates AppScale configuration and state

3. CLOUD FOUNDRY

Cloud Foundry [10] is an open PaaS, which provides choice of clouds, developer frameworks and application services. Cloud Foundry makes application development faster and easier. We can build, test, deploy and scale applications with help of Cloud Foundry. It is an open-source project available through a variety of private cloud distributions and public cloud instances. Cloud Foundry started as a platform to deploy Java Spring applications on Amazon Web Services. VMware acquired the Cloud Foundry and made it into an open-source, multi-language and multi-framework PaaS. Cloud Foundry supports multiple languages and multiple runtimes such as Java, Ruby, Scala, spring and Node.js. Cloud Foundry can run on anything like laptop, desktop, micro cloud, private cloud or public cloud. So, it is called as open PaaS as shown in Figure 2. Cloud Foundry has three dimensions to the platform: choice of frameworks, choice of application services and the deployment choice. Cloud Foundry supports spring for Java, Rails and Sinatra for Ruby, Node.js and JVM languages like Groovy, Grails and Scala. It also supports Microsoft .NET Framework and became the first non-Microsoft platform to support .NET.

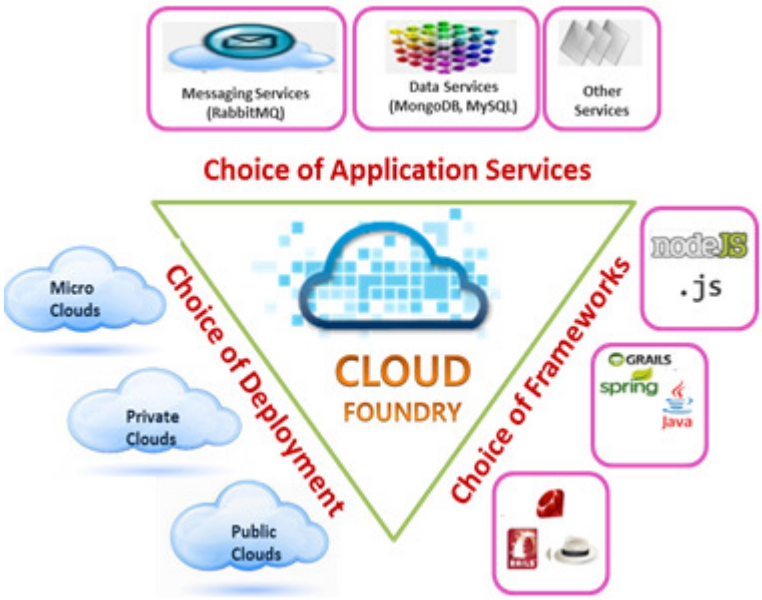

Figure 2. Cloud Foundry as Open PaaS

Cloud Foundry supports RabbitMQ for messaging, MongoDB and Redis for NoSQL, relational databases MySQL and PostgreSQL. Cloud Foundry can be deployed on notebooks through Micro Cloud Foundry. It is the complete version of Cloud Foundry designed to run in a virtual machine. It can also be deployed on Private Cloud or Public Cloud. These features made Cloud Foundry as a flexible PaaS.

Cloud Foundry components perform routing, authentication, messaging, logging, application storage and execution, provide services and take care of application life cycle. The router routes incoming traffic to the appropriate component, usually the Cloud Controller or a running application on a DEA (Droplet Execution Agent) node. The User Account and Authentication (UAA) server work with Login Server to provide identity and authentication management. OAuth2 Server is uses as the user account and authentication server. Cloud controller and health

manager components take care of the application lifecycle in the cloud foundry. Cloud controller is responsible for managing the lifecycle of applications. When a developer pushes an application to cloud foundry, application is targeting the cloud controller. Cloud controller then stores the raw application bits, creates a record to track the application metadata, and directs a DEA node to stage and run the application. Health manager monitor applications to determine their state, version, and number of instances. Applications state may be running, stopped, or crashed. Health manager determine applications expected state, version, and number of instances. It reconciles the actual state of applications with their expected state. Health manager directs the cloud controller to take action to correct any discrepancies in the state of applications. The Droplet Execution Agent manages application instances, tracks, started instances, and broadcasts state messages. Application instances live inside warden containers. Containerization ensures that application instances run in isolation, get their fair share of resources, and are protected from noisy neighbours. Blob Store holds the application code, build packs, and droplets. Applications typically depend on services like databases or third-party SaaS providers. When a developer provisions and binds a service to an application, the service broker for that service is responsible for providing the service instance. Cloud Foundry uses a lightweight publish-subscribe and distributed queueing messaging system for internal communication between components. This internal communication performed via message bus. The metrics collector gathers metrics from the components. Operators can use this information to monitor an instance of Cloud Foundry. The application logging aggregator streams the application logs to the corresponding developers. Cloud Foundry components are shown in Figure 3.

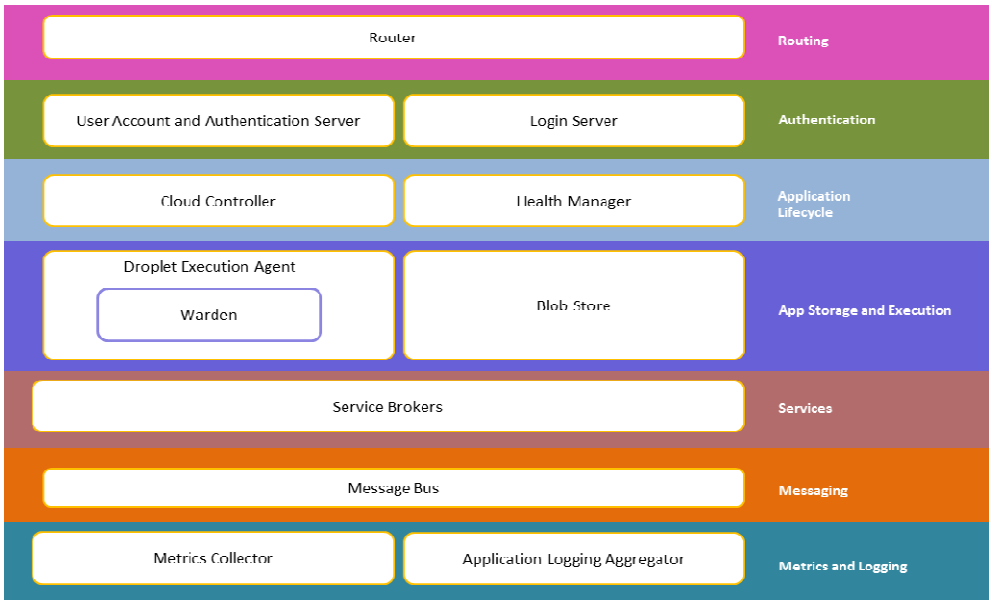

Figure 3. Cloud foundry components

4. CLOUDIFY

Cloudify [11] is another open PaaS cloud application manager. It automates common processes needed to perform and to manage the applications in a cloud environment. Cloudify composed of three main components. The components are Command line interface client, Agents, and Manager. Command line interface client is an executable file which is written in Python. It is

packaged with python and relevant dependencies in an executable file. Command line interface client can run on Windows, Linux and Mac operating systems. Command line interface client preform two tasks. First one is manager bootstrapping and another is managing applications. Bootstrapping is the process of installing the Cloudify manager. Command line interface client provides the user with the full set of functions for deploying and managing applications including log and event browsing.

Cloudify Agents are responsible for managing the manager's command execution using a set of plugins. There is a manager side agent per application deployment and optional agent on each application Virtual Machine (VM). The manager side agents handle IaaS related tasks, like creating a VM or a network, and binding a floating IP to a VM. Manager side agents can also be used with other tools such as REST to remotely execute tasks. The application side agents are optionally located on application VM's. The user can state in the blueprint which VM's will have an agent installed on them. The application side agents are installed by the manager side agent as part of the VM creation task. Once running, the application side agent can install plugins and execute tasks locally. Typical tasks will be middleware installation and configuration, and application modules deployment.

Cloudify Manager deploys and manages applications described in blueprints. The manager's main responsibilities are to run automation processes described in workflow scripts and issue execution commands to the agents. Cloudify is controlled via a REST API. The REST API covers all the cloud orchestration and management functions. Cloudify's Web GUI works with the REST API to add additional value and visibility. Cloudify uses a Workflow engine to allow automation process through built-in and custom workflows. Workflow engine is responsible of timing and orchestrating tasks for creating or manipulating the application components. The user can write custom workflows in Python using API's that provide access to the topology components.

Figure 4. Cloudify Stack

Cloudify uses different databases as data store, some of the technologies for processing and messaging, and different servers as front end. Total stack is shown in Figure 4. Cloudify uses elastic search as its data store for deployment state. The deployment model and runtime data are stored as JSON documents. Blueprints are stored in the elastic search and it is used as runtime DB. Cloudify uses InfluxDB as the monitoring metrics repository. Influx provides flexible schema for metrics and metrics metadata as well as a query language. Cloudify stores every metric reported by a monitoring tool into influxdb and define time based aggregations as well as

statistic calculations. Clodify uses RabbitMQ task broker for messaging. Cloudify offers a policy engine that runs custom policies in order to make runtime decisions about availability, service level agreement, etc. For example, during installation, the policy engine consumes streams of events coming from monitoring probes or tools. The policy engine analyses these streams to decide if a specific node is up and running and provides the required functionality. Policies are registered, activated, deactivated and deleted by the Workflow Engine. For logging purpose logstash is used and agent play main role in processing. Nginx proxy and file server, Flask or Gunicorn REST server, and Node.js GUI servers can be used as front end in the Cloudify.

5. OPEN SHIFT

OpenShift [12] enables us to create, deploy and manage applications within the cloud. Two basic functional units of the Openshift are the Broker and Node servers. Communication between the Broker and Nodes is done through a message queuing service. Broker is the single point of contact for all application management activities. It is responsible for managing user logins, DNS, application state, and general orchestration of the applications. Customers don't contact the broker directly; instead they use the Web console or CLI tools to interact with Broker over a REST based API. Nodes are the systems that host user applications. In order to do this, the Nodes are configured with Gears and Cartridges. A gear represents the part of the Node's CPU, RAM and base storage that is made available to each application. An application can never use more of these resources allocated to the gear, with the exception of storage. OpenShift supports multiple gear configurations, enabling users to choose from the various gear sizes at application setup time. When an application is created, the Broker instructs a Node to create a new gear to contain the application. Cartridges represent pluggable components that can be combined within a single application. These include programming languages, database engines, and various management tools. Users can choose from built-in cartridges that are served directly through OpenShift, or from community cartridges that can be imported from a git repository. The built-in cartridges require the associated languages and database engines to be installed on every Node.

6. OTHER PAAS

In this section we are going to give brief introduction about Stratos, Stakato and mOSAIC open PaaS environments.

6.1. Stratos

Apache Stratos [5] is a highly-extensible PaaS framework that helps to run Apache Tomcat, PHP, and MySQL applications, and can be extended to support many more environments on all major cloud infrastructures. For developers, Stratos provides a cloud-based environment for developing, testing, and running scalable applications. In Single JVM deployment model Stratos could accommodate up to 100 cartridge instances. In a distributed deployment model Stratos could accommodate up to 1000 cartridge instances.

6.2. Stakato

Stackato [6] is open PaaS software based on Cloud Foundry, Docker and other open-source components. It has multi-tenancy capabilities and can be installed on internal infrastructure or public cloud. Multi-tenancy capabilities are important because they allow us to run multiple

applications on the same IaaS infrastructure. Stackato allows developers to automatically package applications into their own Docker containers and scales instances up or down on demand. Stackato provisions all required components, including languages, frameworks and service bindings, automates logging and monitoring, allows for automated application versioning and rollback.

6.3. mOSAIC

mOSAIC [9] is an open-source API and platform for designing and developing multi-Cloud-oriented applications. The architecture has been designed with open and standard inter faces. The main goal is to provide a unified cloud programming interface which enables flexibility to build applications across different cloud providers. The main middleware components providing integration features are the Cloudlet, Connector, Interoperability, and Driver API. 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. 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.

7. COMPARISON OF MAJOR PAAS

This section compares the major open PaaS frameworks. Table 2 shows the basic functionality and its corresponding AppScale, Cloud Foundry, Cloudify, and OpenShift architectural components.

Table 2. Open PaaS Components comparison

Functionality	**AppScale**	**Cloud Foundry**	**Cloudify**	**OpenShift**
Core functionality	AppController	Cloud controller	Manager	Broker
Providing third party database services	Database Master	Service Broker	Agent	Cartridge
Routing of incoming traffic	AppLoadBalancer	Router	Manager	REST API
Querying the state of apps	AppScale Tools	Cloud controller	CLI client	Broker
Messaging	AppController	Message Bus	Manager	Broker
Application instance management	AppServer	Droplet Execution Agent	Agent	Node
Application state change	AppLoadBalancer	Health Manager	Manager	Broker
Containerization	Database Slave	Warden	Agent	Gear
Load balancing of user requests	AppLoadBalancer	Droplet Execution Agent	Manager	Broker
Framework provider	AppServer	Blob Store	Agent	Cartridge

Table 3 shows the AppScale, Cloud Foundry, Cloudify, and OpenShift PaaS supported languages (java, python, ruby), databases (MongoDB, MySQL, HBase) and frameworks (spring, rails, and flask). In OpenShift, languages and databases are supported in the form of cartridges. User defined cartridges are also allowed in OpenShift. Cloud Foundry provisions languages in the

form of build packs. Users can also pick to write their own build packs. Cloudify, Cloud Foundry and Openshift have extensible language support feature.

Table 3. Language, Database and Frameworks supported by open PaaS

	Languages	**Databases**	**Frameworks**
AppScale	Python, Java, Go, PHP	Cassandra, HBase, Hypertable, MongoDB, SimpleDB, MySQL	Django, Flask, Spring
Cloud Foundry	Java, Ruby, Scala, Node.js, Groovy, Grails, PHP, Go, Python	MonogoDB, MySQL, PostgreSQL	Spring, Rails, Grails, Play, Sinatra
Cloudify	Java, PHP, Ruby	MySQL, MongoDB	-
OpenShift	Java, PHP, Ruby, Python, Perl, JavaScript, Node.js	PostgreSQL, MySQL, MongoDB	Rails, Flask, Django, Drupal, Vert.x

Table 4 shows the features support by AppScale, Cloud Foundry, Cloudify, and OpenShift platforms.

Table 4. Open PaaS Considerable Feature Support

Features	**AppScale**	**Cloud Foundry**	**Cloudify**	**OpenShift**
Relational database support	Yes	Yes	Yes	Yes
NoSQL database support	Yes	Yes	Yes	Yes
Horizontal Scaling	Yes	Yes	Yes	Yes
Vertical Scaling	No	Yes	No	Yes
Auto Scaling	Yes	No	Yes	Yes
Spring Framework support	Yes	Yes	No	No

8. CONCLUSIONS

Cloud computing service models like Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) are introduced in this paper. PaaS is explained in detail with the help of open PaaS packages like AppScale, Cloud Foundry, Cloudify, and OpenShift. AppScale components are explained in table format, Cloud Foundry components are explained in detailed with a diagram, Cloudify and OpenShift components are also explained. Stakato, Stratos and mOSAIC open PaaS environments also explained in this paper. Comparative study is performed among the AppScale, Cloud Foundry, Cloudify and OpenShift open PaaS componets.

ACKNOWLEDGEMENTS

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education. (Grant Number: 2011-0011507).

REFERENCES

[1] Daniel Nurmi, Richard Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff & Dmitrii Zagorodnoy, (2009) "The Eucalyptus Open-Source Cloud-Computing System", CCGrid, pp124-131.

[2] Bunch, C., Chohan, N., Krintz, C., Chohan, J., Kupferman, J. & Lakhina, P., et al., (2010) "An Evaluation of Distributed Datastores Using the AppScale Cloud Platform", IEEE International Conference on Cloud Computing.

[3] Bunch, Chris, Navraj Chohan & Chandra Krintz, (2011) "Appscale: open-source platform-as-a-service", UCSB Technical Report.

[4] Varma, N. M. K., Min, D. & Choi, E. (2011) "Diagnosing CPU utilization in the Xen virtual machine environment", In Computer Sciences and Convergence Information Technology (ICCIT), 6th International Conference, pp. 58-63, IEEE.

[5] Pawluk, P., Simmons, B., Smit, M., Litoiu, M. & Mankovski, S. (2012) "Introducing STRATOS: A cloud broker service", In 2012 IEEE Fifth International Conference on Cloud Computing, pp. 891-898, IEEE.

[6] Fortiş, T. F., Munteanu, V. I., & Negru, V., (2012), "Towards a service friendly cloud ecosystem", In Parallel and Distributed Computing (ISPDC), 11th International Symposium, pp. 172-179, IEEE.

[7] Hossny, E., Khattab, S., Omara, F. & Hassan, H., (2013) "A Case Study for Deploying Applications on Heterogeneous PaaS Platforms", In Cloud Computing and Big Data (CloudCom-Asia), International Conference on (pp. 246-253), IEEE.

[8] Varma, N. M. K. & Choi, E., (2013) "Extending Grid Infrastructure Using Cloud Computing", In Ubiquitous Information Technologies and Applications, pp. 507-516, Springer Netherlands.

[9] Marpaung, J., Sain, M. & Lee, H. J., (2013) "Survey on middleware systems in cloud computing integration", In Advanced Communication Technology (ICACT), 15th International Conference, pp. 709-712, IEEE.

[10] D. Bernstein, (2014) "Cloud Foundry Aims to Become the OpenStack of PaaS", IEEE Cloud Computing, (2), 57-60.

[11] Graham, S. T. & Liu, X., (2014) "Critical evaluation on jClouds and Cloudify abstract APIs against EC2, Azure and HP-Cloud", In Computer Software and Applications Conference Workshops (COMPSACW), IEEE 38th International, pp. 510-515, IEEE.

[12] A. Lomov, (2014) "OpenShift and Cloud Foundry PaaS: High-level Overview of Features and Architectures", Available at www.altoros.com/openshift_and_cloud_foundry_paas.html.

AUTHORS

Name: **Mohan Krishna Varma Nandimandalam**
Address: B-304, DIS Lab, School of Business IT, International Building,
Kookmin University, Seoul-136702, South Korea
Education: Completed Bachelor of Computer Applications degree in 2002, Received Master of Science in Information Systems degree in 2004 and Master of Technology in Computer Science and Engineering in 2007 from VIT University, India. At present studying Ph.D. in Graduate School of Business IT, Kookmin University, South Korea.

Eunmi Choi is a Professor in the School of Business IT,
Chairperson of School of Management Information Systems,
Head of Distributed Information System & Cloud Computing Lab., and
Executive Chief of Business IT Graduate School at Kookmin University, Korea,
Her current research interests include big data infra system and processing, cloud computing, cyber physical system, information security, distributed system, SW meta-modelling, and grid & cluster computing. Professor Choi received and MS and PhD in computer science from Michigan State University, U.S.A., in 1991 and 1997, respectively, and BS in computer science from Korea University in 1988.