

# COMPARATIVE STUDY OF VARIOUS PLATFORM AS A SERVICE FRAMEWORKS

Mohan Krishna Varma and Eunmi Choi\*

DISC Lab, Graduate School of Business IT, Kookmin University, Seoul, South Korea

\*Corresponding Author, School of Business IT, Kookmin University, Seoul, Korea

## ABSTRACT

Cloud computing is an emerging paradigm 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 different kinds of PaaS frameworks. PaaS model provides choice of cloud, developer framework and application service. In this paper, detailed study of four open PaaS frameworks like AppScale, Cloud Foundry, Cloudify, and OpenShift are explained with the architectural components. We also explained more PaaS packages like Stratos, mOSAIC, BlueMix, Heroku, Amazon Elastic Beanstalk, Microsoft Azure, Google App Engine and Stakato briefly. In this paper we present the comparative study of PaaS frameworks.

## KEYWORDS

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

## 1. INTRODUCTION

Cloud computing [2] is a widely used technology 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) [8] and Infrastructure as a Service (IaaS) [9]. Cloud computing service models are shown in Figure 1. Customer Relationship Management (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 [3], 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.



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 the applications. This paper deals with the architectural components of major open PaaS frameworks like AppScale, Cloud Foundry, Cloudify and OpenShift. And also given some explanation about PaaS frameworks such as Stratos, mOSAIC, BlueMix, Heroku, Amazon Elastic Beanstalk, Microsoft Azure, Google App Engine and Stakato. The paper is organized as follows. Section 2 introduces 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 PaaS frameworks are introduced in Section 6, comparison of PaaS frameworks are given in Section 7 and finally Section 8 concludes the paper.

## 2. APPSCALE

AppScale [4] is a scalable, distributed, and fault-tolerant cloud runtime system that executes over cluster resources. It can be deployed on Xen [5], 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

<b>Component</b>	<b>Language</b>	<b>Functionality</b>
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 [13] 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.

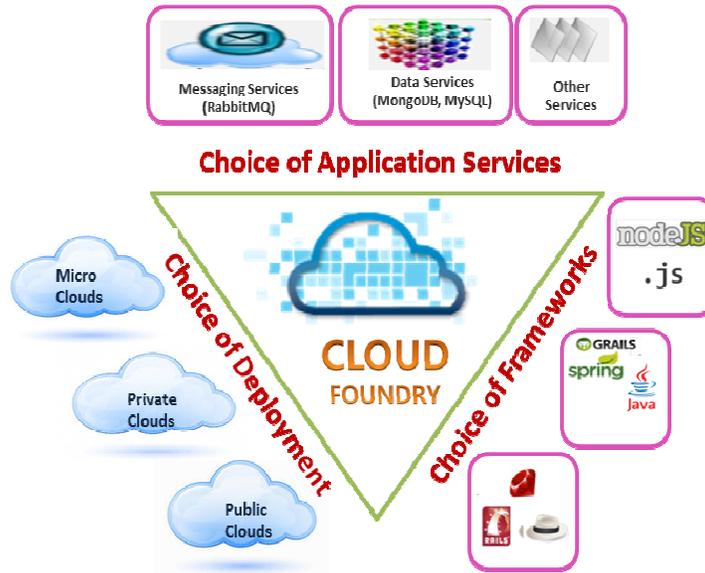


Figure 2. Cloud Foundry as Open PaaS

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.

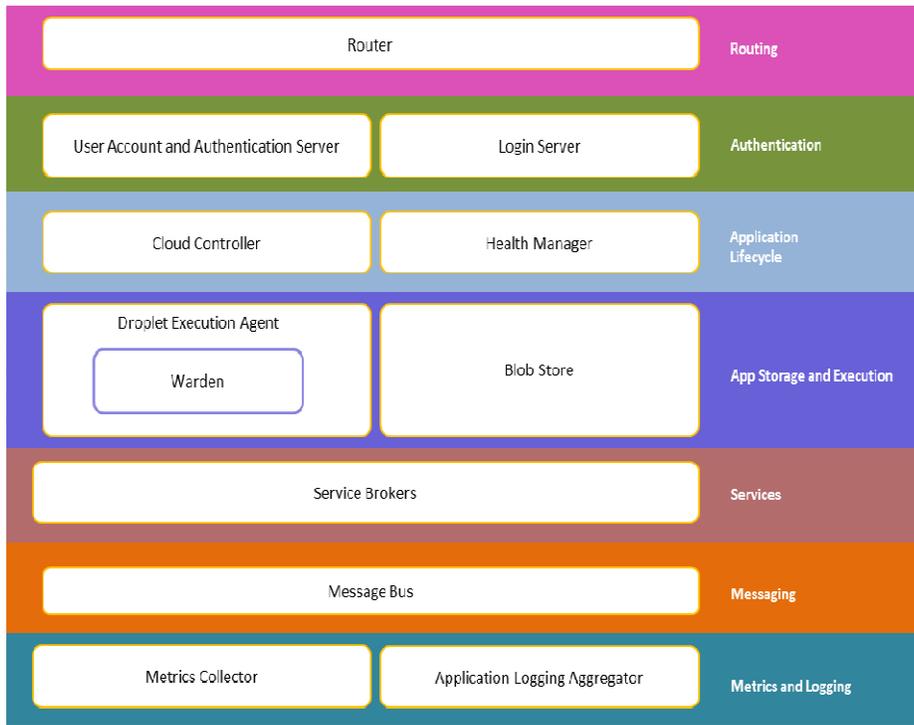


Figure 3. Cloud foundry components

#### 4. CLOUDIFY

Cloudify [14] 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 perform 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.

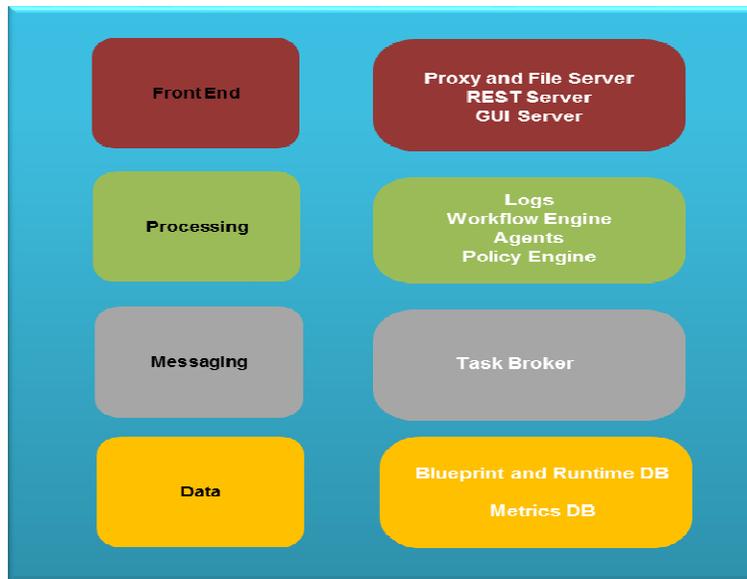


Figure 4. Cloudify Stack

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. Cloudify 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 [15] 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. The division is shown in Figure 5.

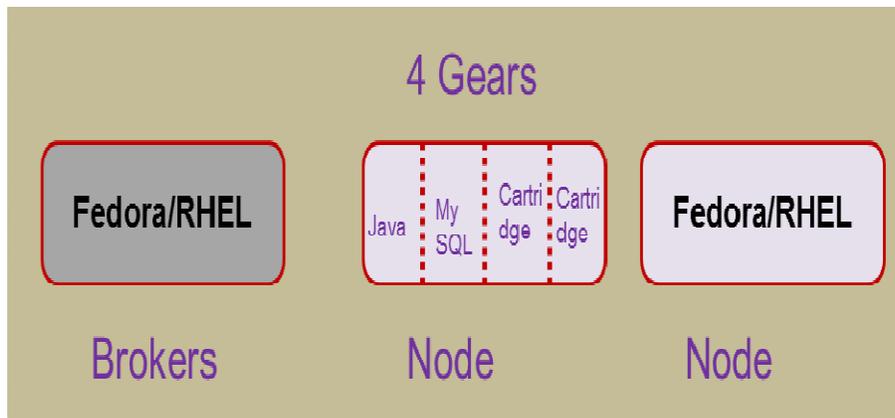


Figure 5. OpenShift

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. MORE PAAS FRAMEWORKS

In this section we are going to give brief introduction about Stratos, mOSAIC, IBM BlueMix, Heroku, Amazon Elastic Beanstalk, Microsoft Azure and Google App Engine, and Stakato PaaS frameworks.

### **6.1. STRATOS**

Apache Stratos [6] 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. MOSAIC**

mOSAIC [10] is an open-source API and platform for designing and developing multi-Cloud-oriented applications. The architecture has been designed with open and standard interfaces. 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.

### **6.3. BLUEMIX**

Bluemix [16] is the newest cloud offering from IBM. BlueMix allows developers to create, deploy, and manage applications on the cloud. It is an implementation of IBM's Open Cloud Architecture based on Cloud Foundry, an open source Platform as a Service (PaaS). It delivers enterprise-level services that can easily integrate with cloud applications without requiring to know how to install or configure them.

### **6.4. HEROKU**

Heroku [17] is a PaaS solution created on top of the Amazon EC2 that allows to deploy, run and manage applications written in Ruby, Node.js, Java, Python, Clojure, Scala and PHP. It runs on top of Amazon EC2 VMs and offers a readymade set up environment in which the developer can upload code, compile it and run it. All these things can be done with some simple commands. The developer has the opportunity to test and deploy applications without necessary knowledge of underlying infrastructure. Heroku can manage the resources required to scale the project.

### **6.5. ELASTIC BEANSTALK**

Amazon Elastic Beanstalk [12] supports users to deploy their applications on Amazon Web Service (AWS) cloud platform. AWS Elastic Beanstalk PaaS framework allows users to create applications and push them to a definable set of AWS services, including Amazon EC2, Amazon S3, Amazon Simple Notification Service (SNS), Amazon CloudWatch, auto scaling, and elastic load balancers. Elastic Beanstalk frameworks support different languages for cloud development.

## 6.6. AZURE

Microsoft Windows Azure [11] allows the application deployment and management on Azure cloud platform. The Microsoft Azure framework offers a cloud services operating system and a set of services to support easy development and operation of applications for cloud platform. Azure provides functionality to build and manage applications that span from consumer Web to enterprise scenarios.

## 6.7. APP ENGINE

Google App Engine [18] allows developers to deploy their application on Google cloud platform. App Engine framework allows developers to use Java, Python and some other languages for developing their web applications.

## 6.8. STAKATO

Stackato [7] 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.

## 7. COMPARISON OF MAJOR PAAS

In this section major PaaS frameworks comparison is given in the table format. 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, OpenShift, Apache Stratos, mOSAIC, IBM BlueMix, Heroku, Amazon Elastic Beanstalk, Microsoft Azure and Google App Engine PaaS environment's supported languages (java, python, ruby, perl, groovy), databases (MongoDB, MySQL, HBase, PostgreSQL), frameworks (spring, rails, and flask), and deployments (private or public). 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. AppScale, Cloud Foundry, Cloudify, OpenShift, Apache Stratos and mOSAIC are open source cloud platforms, for this reason their deployment is given as private PaaS. IBM BlueMix, Heroku, Amazon Elastic Beanstalk, Microsoft Azure and Google App Engine PaaS environments can be deployed on public cloud and these are not open source, so these things are considered as Public PaaS.

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

PaaS	Languages	Databases	Frameworks	Deployment
<b>AppScale</b>	Python, Java, Go, PHP	Cassandra, HBase, Hypertable, MongoDB, SimpleDB, MySQL	Django, Flask, Spring	Private PaaS
<b>Cloud Foundry</b>	Java, Ruby, Scala, Node.js, Groovy, Grails, PHP, Go, Python	Monogodb, MySQL, PostgreSQL	Spring, Rails, Grails, Play, Sinatra	Private PaaS
<b>Cloudify</b>	Java, PHP, Ruby	MySQL, MongoDB	Not supported	Private PaaS
<b>OpenShift</b>	Java, PHP, Ruby, Python, Perl, JavaScript, Node.js	PostgreSQL, MySQL, MongoDB	Rails, Flask, Django, Drupal, Vert.x	Private PaaS
<b>Stratos</b>	Java, PHP	MySQL	Not supported	Private PaaS
<b>mOSAIC</b>	Java	Not supported	Not supported	Private PaaS
<b>BlueMix</b>	PHP, Python, Ruby, Java, Node.js, Go	Cleardb, Elephantsql	Sinatra, Rails	Public PaaS
<b>Heroku</b>	Scala, Ruby, Python, PHP, Node.js, Java, Groovy, Go, Clojure	PostgreSQL	Django, Flask, Rails, Grails, Play	Public PaaS
<b>Amazon Elastic Beanstalk</b>	.Net, Go, Java, Node.js, PHP, Python, Ruby	Not supported	Not supported	Public PaaS
<b>Microsoft Azure</b>	Java, .Net, Node.js, Ruby, Python, PHP	Not supported	CakePHP, Django	Public PaaS
<b>Google App Engine</b>	Go, Java, PHP, Python	Not supported	Django	Public PaaS

Table 4 compares the relational database support, NoSQL Support, horizontal scaling, vertical scaling, auto scaling, spring framework support, Add-ons and language extensibility features of AppScale, Cloud Foundry, Cloudify, OpenShift, Apache Stratos, mOSAIC, IBM BlueMix, Heroku, Amazon Elastic Beanstalk, Microsoft Azure and Google App Engine PaaS Frameworks. The Table 4 shows “yes” for supported features and “no” for the unsupported features in the corresponding framework columns.

Table 4. PaaS framework’s Considerable Feature Support

Features	AppScale	Cloud Foundry	Cloudify	OpenShift	Stratos	mOSAIC	BlueMix	Heroku	Elastic Beanstalk	Microsoft Azure	App Engine
Relational database support	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No	No	No
NoSQL database support	Yes	Yes	Yes	Yes	No	No	No	No	No	No	No
Horizontal Scaling	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Vertical Scaling	No	Yes	No	Yes	No	No	Yes	Yes	Yes	Yes	No
Auto Scaling	Yes	No	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
Spring Framework support	Yes	Yes	No	No	No	No	No	No	No	No	No
Add-ons support	No	No	No	No	No	No	Yes	Yes	No	Yes	Yes
Language Extensibility	No	Yes	Yes	Yes	Yes	No	Yes	Yes	No	Yes	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, Cloudify and OpenShift components are explained in detailed with a diagrams. Apache Stratos, mOSAIC, IBM BlueMix, Heroku, Amazon Elastic Beanstalk, Microsoft Azure, Google App Engine, and Stakato PaaS frameworks are also explained in this paper. Comparative study is performed among the AppScale, Cloud Foundry, Cloudify and OpenShift open PaaS componets. Supported language, databases, frameworks, and deployments like public PaaS or Private PaaS are also listed for AppScale, Cloud Foundry, Cloudify, OpenShift, Apache Stratos, mOSAIC, IBM BlueMix, Heroku, Amazon Elastic Beanstalk, Microsoft Azure and Google App Engine PaaS environments. Relational database support, NoSQL Support, horizontal scaling, vertical scaling, auto scaling, spring framework support and language extensibility features are compared among the AppScale, Cloud Foundry, Cloudify, OpenShift, Apache Stratos, mOSAIC, IBM BlueMix, Heroku, Amazon Elastic Beanstalk, Microsoft Azure and Google App Engine frameworks.

## 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 Grzegorzczak, Graziano Obertelli, Sunil Soman, Lamia Youseff & Dmitrii Zagorodnoy, (2009) "The Eucalyptus Open-Source Cloud-Computing System", CCGrid, pp124-131.
- [2] Rimal, B. P., Choi, E., & Lumb, I., (2009) "A taxonomy and survey of cloud computing systems", Fifth International Joint Conference on INC, IMS and IDC, pp. 44-51, IEEE.
- [3] 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.
- [4] Bunch, Chris, Navraj Chohan & Chandra Krintz, (2011) "Appscale: open-source platform-as-a-service", UCSB Technical Report.
- [5] 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.
- [6] 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.
- [7] Fortiș, 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.
- [8] 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, pp. 246-253, IEEE.
- [9] Varma, N. M. K. & Choi, E., (2013) "Extending Grid Infrastructure Using Cloud Computing", In Ubiquitous Information Technologies and Applications, pp. 507-516, Springer Netherlands.
- [10] 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.
- [11] Lima, R. C., Hollanda Filho, R., Sampaio, A., & Mendonca, N., (2013) "TrexCloud: Java EE IaaS Cloud Deployment Made Easy", Sixth International Conference on Cloud Computing, pp. 470-477, IEEE.
- [12] Akinbi, A., Pereira, E., & Beaumont, C., (2013) "Evaluating security mechanisms implemented on public Platform-as-a-Service cloud environments case study: Windows Azure", Internet Technology and Secured Transactions (ICITST), 8th International Conference, pp. 162-167, IEEE.
- [13] D. Bernstein, (2014) "Cloud Foundry Aims to Become the OpenStack of PaaS", IEEE Cloud Computing, (2), 57-60.
- [14] 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 conference, pp. 510-515, IEEE.
- [15] 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](http://www.altoros.com/openshift_and_cloud_foundry_paas.html).
- [16] Dettori, P., Frank, D., Seelam, S. R., & Feillet, P., (2014) "Blueprint for Business Middleware as a Managed Cloud Service", In Cloud Engineering (IC2E), IEEE International Conference, pp. 261-270, IEEE.
- [17] Chenaru, O., Stanciu, A., Popescu, D., Sima, V., Florea, G., & Dobrescu, R., (2015), "Open cloud solution for integrating advanced process control in plant operation", In Control and Automation (MED), 23rd Mediterranean Conference, pp. 973-978, IEEE.
- [18] Ferriman, B., Hamed, T., & Mahmoud, Q. H., (2015) "Storming the cloud: A look at denial of service in the Google App Engine", Computing, Networking and Communications (ICNC), International Conference, pp. 363-368, IEEE.

## AUTHORS

**Mohan Krishna Varma Nandimandalam** is Completed Bachelor of Computer Applications degree in 2002 and Received Master of Science in Information Systems degree in 2004 from Sri Venkateswara University, Tirupati, Andhra Pradesh, India, and Master of Technology in Computer Science and Engineering in 2007 from VIT University, India. Currently working as a full time researcher in Distributed information Systems and Cloud computing lab and doing 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.

