MagpieOS

A Native Cloud OSGi Platform

White Paper

Release February 2016

Written by
Natthaphong Ruengpanyawut
Platform Architect

Reviewed by
Sutthichai Taetong
Pornchai Tipwataksorn
Core Maintainers

Table of Contents

Vision

What is MagpieOS?

In the most summarized term, we can call MagpieOS as a *"Native Cloud OSGi Platform"*. Although it is originated from the motivation of trying to solve the common challenges of SaaS software in *Business Solution* fields, we strongly believe that it can be used as an underlying platform for any kinds of Cloud software since it is an OSGi compliance platform.

Common Cloud (SaaS) Software Challenges

	Topic	Solving Techniques	Solution Providers
1)	**Sharing Pool Resources** (to minimize the infra investment and operating costs)	- Virtualization (using IaaS providers as service underlying infra)	- AWS - Azure - Google Compute Engine - Rackspace - Digital Ocean - etc.
2)	**Execution Isolation** (an execution from one tenant will not influence other tenant resources)	- Isolation - Container	- Docker Container - etc.
3)	**Scalability** (to handle large amount of users)	- Clustering - Load Balancing	- Docker Swarm - Apache ACE - etc.
4)	**Elasticity** (scale up / down resources as needed to minimize service cost—or saying that to maximize the utilization)	- Provisioning	- AWS CloudFormation - Docker Machine - etc.
5)	**Customization** (how a SaaS software using by all users from multi-tenant running on the same SaaS provider server/infra can be customized to fit the user needs individually)	- Isolation (allowing the user to customize their software or system without influencing on others)	- Docker Container - etc.
6)	**Service Extensibility** (how a SaaS software can be easily extended to serve the future needs from tremendous user growth and use cases?)	- Modularity	- OSGi (Apache Felix) - etc.

7) **Service Dynamicity** (how software components can be installed to / uninstalled from the system in Plug & Play manner—like installing an app from Play / App Store—without interrupting the system?)	- Modularity (especially for a framework which promotes service dynamicity)	- OSGi (Apache Felix) - etc.
8) **Easy and Seamless App / Data Integration** (how could we easily integrate the data from various services into one system as a whole?)	- API	- SaaS / PaaS Specific

Table-1: The common challenges of Cloud (SaaS) Software

Remarks:

- We only list the name of Solutions Providers which are intended to be used as the underlying technologies of MagpieOS.
- We may categorize the topics above into two categories:
    - Cloud Technology Topics: addressed in (1) – (5)
    - Software Architecture Topics: addressed in (6) – (8)

  Since SaaS is "Cloud + Software" delivered as a service, so we may count the union set of both categories as a single set of Common SaaS Software Challenges.
- For the topics (5) – (8), they are the basic needs of—but not limited to—business solution software.
- Although there are already the solution providers in most of all topics listed above, as a software developer who should be focusing on developing software features, learning to use all of those tools, platforms, or technologies is not a trivial task to do. We intended to make MagpieOS to be a platform helping developers to be able to focus on their apps.
- There may be some other currently well-known Solution Providers for each topic which are not listed in the table such as Heroku—a comparable technology to Docker. So, we put the explanation of why we decided to use this rather than that in *Platform Architecture* section.
- If you are having a good understanding in the concepts relating to MagpieOS but still questioning that "Are we trying to reinvent the wheel?", you can skip reading the following section (Use Case Scenario) and jump into *Business Analysis* section.

Glossary

	Term	Description
1)	**Magpie**	If you use this term as a single word, it usually means Magpie Platform and represents both its architecture and ecosystem.
2)	**MagpieOS**	In the early stage of developing Magpie Platform, we know that using this term is not technically correct. According to the definition on Wikipedia, an *Operating System (OS)* is a software managing the hardware[*]. But, we use this term to envision the Magpie Platform foreseen in the *Internet of Things (IoT)* world. In other words, for technically, if we have Magpie Instance running on the hardware which is not hosted on the Cloud—especially for running on a single standalone hardware (single NestVM) i.e. RPi or Arduino, then we can call that NestVM as an OS if and only if we provide APIs to Magpie App developers to control and interact with the hardware (like Android Platform do). We will be greatly benefited from OSGi modularity on its `services` principle since—as an app developer—you are not required to be aware of what hardware that your app will be run on. Only thing you need to consider is to consume the service you need and let Magpie to tell the user that your app can be installed on their Magpie Instance or not.
3)	**Magpie Instance**	Magpie Instance represents a logical view of a single device / server to the users. A single Magpie Instance may be backed by one or multiple NestVMs. We use the word "device" (not only "server") since we aimed MagpieOS can be run outside the Cloud [see (2) for more details].
4)	**NestVM**	It's a JVM pre-installed with OSGi Framework and platform low-level bundles. When we refer to NestVM, we only mean a single JVM.
5)	**Magpie Module**	A software package in which its structure inherits OSGi `bundle`.
6)	**Magpie Package**	A set of Magpie Modules. Its concept is inherited from OSGi `feature`.
7)	**Magpie App**	Magpie App is a Magpie Module or Magpie Package which provides the UI or ready-to-use functionalities to the end users. It can be published on Magpie App Store
8)	**Magpie Repository**	It's an OSGi `repository` which stores Magpie Modules & Magpie Packages published by Magpie developers. It is mainly used as centralized online storage for software libraries.
9)	**Magpie App Store**	It's a special type of Magpie Repository which only stores Magpie Apps.

Table-2: Glossary

[*] Operating System – Wikipedia, *https://en.wikipedia.org/wiki/Operating_system*

Goal & Key Features

Our main goal is to bring MagpieOS in reality. It is aimed to serve the visionary use cases shown in above section. To summarize them, we conclude its key features into the list below.

MagpieOS Key Features

	Key Feature	Description
1)	**Application Centric**	The main goal for all PaaS is letting developers mainly focus on their application. It will help developers on hiding the complexities in designing distributed system running on cluster-enabled infrastructures.
2)	**Plug & Play, not Rebuild**	We believe that service dynamicity derived from OSGi framework is a key. Like using a smartphone, when we want to enhance or change its features, we download, install, use, and uninstall an app in plug & play manner. We have never rebuilt the entire system as we just only need installing an app.
3)	**App / Data Integration as a Whole**	Empowering by OSGi runtime, developers will be allowed to use or consume other apps / modules services in OSGi way—easily call Java interface method from the source code directly. No more unnecessary RESTful API calls to the remote servers. This enables us to automatically store various apps data on the same Magpie Instance as a whole single system, not distributed to various SaaS depending infrastructures. More importantly, from the users' point of view, they will see their individual Magpie Instance as a whole system with multiple collaborating applications installed behind the scene.
4)	**YOU run an App, not ME**	MagpieOS App Store model is likely to nowadays smartphone's model—unlike many current PaaS models i.e. Google App Engine. From our perspective, an app is a software package which is developed and will be shipped to be installed on end user instances remotely, not running on developer-own instances.
5)	**Modularity by Design**	Since MagpieOS is ran on OSGi runtime, this brings us a great benefit from the modularity world: a REUSABLE software design. Developers can build their apps / modules on top other apps / modules developed by anyone else by declaring normal OSGi DEPENDENCIES, then start using the others' services in OSGi conventional way.
6)	**Ship Apps Everywhere**	MagpieOS provides a public hosted application market place allowing end users to purchase and install Magpie Apps into their Magpie Instance instantly. This helps developers on distributing their apps via a centralize app portal.
7)	**Not only for the WEB**	As MagpieOS is an OSGi compliance platform, it can run any OSGi components—we called it `bundles`—on the Cloud.
8)	**Host your own Infra**	MagpieOS provides the server-side software distributions letting users to run Magpie Platform on their own infrastructures.
9)	**Abundant Developer Resources**	Using Java as its programming language—which is one of the most popular programming language[*]—provides us tremendous resources available on the Internet for free!

Table-3: MagpieOS Key Features

[*] The 2015 Top Ten Programming Languages – IEEE Spectrum, *http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages*

Business Analysis

Are we trying to reinvent the wheel?

Of course not, therefore, there are several technologies offering similar features to MagpieOS such as Google App Engine, heroku.com, force.com, mendix.com etc., there are still some characteristics which make MagpieOS different. In this topic, we'll navigate you through our currently known comparable technologies and tell you the KEY DIFFERENCES to MagpieOS making it still be an essential technology for the future.

We will use the comparison check-list table to show you that how MagpieOS different from other PaaS services. Its intention is to give the reasons why we have to invent MagpieOS if we need these set of features—not critiquing on them. If you are not clear about what actually each *Key Feature* means, you could go back to read *Goal & Key Features* first.

Google App Engine

Comparison Check-list:

	Key Feature	Checked	Remarks
1)	Application Centric	√	
2)	Plug & Play, not Rebuild	X	
3)	App / Data Integration as a Whole	X	
4)	YOU run an App, not ME	X	Google App Engine Apps must be run on developer-own account, not end-user-own account.
5)	Modularity by Design	X	
6)	Ship Apps Everywhere	√	Google Apps Marketplace
7)	Not only for the WEB	X	
8)	Host your own Infra	X	
9)	Abundant Developer Resources	√	Google App Engine supports various of programming languages not only for Java.

Similarity Score: 3 / 9

Notes:
N/A

Heroku.com

Comparison Check-list:

	Key Feature	Checked	Remarks
1)	Application Centric	√	
2)	Plug & Play, not Rebuild	X	
3)	App / Data Integration as a Whole	X	From the end-user point of view, Heroku does not provide any mechanism to let user select and install

		multiple apps into their owned instances. Its add-on mechanism is for developer's purposes.
4) YOU run an App, not ME	X	Although Heroku provides add-on functionality, the key issue that we assert this feature as false is the fact that a Heroku app must be run in your (developer-own) dynos, not the end-user-own dynos.
5) Modularity by Design	√	Heroku Add-ons
6) Ship Apps Everywhere	√	Heroku Add-ons
7) Not only for the WEB	X	Heroku only open HTTP / HTTPS port to the outside world.
8) Host your own Infra	X	
9) Abundant Developer Resources	√	Heroku supports various of programming languages.

Similarity Score: 4 / 9

Notes:
N/A

Force.com

Comparison Check-list:

Key Feature	Checked	Remarks
1) Application Centric	√	
2) Plug & Play, not Rebuild	√	
3) App / Data Integration as a Whole	√	
4) YOU run an App, not ME	√	An app is installed into end-user accounts. So, the developers has no direct variable costs on providing an app to their customers.
5) Modularity by Design	√	
6) Ship Apps Everywhere	√	Salesforce.com AppExchange
7) Not only for the WEB	X	
8) Host your own Infra	X	
9) Abundant Developer Resources	X	Force.com app uses its own programming language: Apex, which is not still ranked in the top 10 of the most popular programming languages of the world[*]. Although it has its own active community which may be sufficient if you are developing an app running on force.com, its developer resources: software libraries, tools etc., are not quantitative comparable to other leading programming languages such as C-family, Java, python, php, Ruby, etc.

Similarity Score: 6 / 9

Notes:
N/A

[*] The 2015 Top Ten Programming Languages – IEEE Spectrum, *http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages*

Mendix.com

Comparison Check-list:

Key Feature	Checked	Remarks
1) Application Centric	√	
2) Plug & Play, not Rebuild	X	As an end user, you have to install an app in your modeler, then, deploy & run it on your server instance. This is not like Plug & Play manner because it requires you to rebuild & redeploy the system.
3) App / Data Integration as a Whole	√	You can download, build and deploy multiple apps in one server instance.
4) YOU run an App, not ME	√	The users run an app in their own server instance.
5) Modularity by Design	X	We haven't seen any clearly stated documentation about app / service dependency in mendix platform.
6) Ship Apps Everywhere	√	Mendix.com App Store
7) Not only for the WEB	X	
8) Host your own Infra	√	Mendix.com also offers on-premise model.
9) Abundant Developer Resources	√	Mendix.com platform uses Java as its programming language.

Similarity Score: 6 / 9

Notes:

Product Differentiation

To summarize the differentiation among current PaaS providers, we construct the comparison check-list against each PaaS in the table below.

Key Feature	MagpieOS	App Engine	Heroku	Force.com	Mendix
1) Application Centric	√	√	√	√	√
2) Plug & Play, not Rebuild	√	X	X	√	X
3) App / Data Integration as a Whole	√	X	X	√	√
4) YOU run an App, not ME	√	X	X	√	√
5) Modularity by Design	√	X	√	√	X
6) Ship Apps Everywhere	√	√	√	√	√
7) Not only for the WEB	√	X	X	X	X
8) Host your own Infra	√	X	X	X	√
9) Abundant Developer Resources	√	√	√	X	√
Similarity Score		3 / 9	4 / 9	6 / 9	6 / 9

MagpieOS Platform Architecture