Characterizing PaaS Solutions Enabling Cloud Federations

Tamas Pflanzner[1], Roland Tornyai[1,2], Laszlo Szilagyi[2], Akos Goracz[1] and Attila Kertesz[1]

[1]*University of Szeged, Hungary*

[2]*Ericsson Hungary Ltd., Hungary*

ABSTRACT

Cloud Computing has opened new ways of flexible resource provisions for businesses to migrate IT applications and data to the Cloud to respond to new demands from customers. Recently, this form of service provision has become hugely popular, with many businesses migrating their IT applications and data to the Cloud to take advantage of the flexible resource provision that can bring benefits to businesses by responding quickly to new demands from customers. Cloud Federations envisage a distributed, heterogeneous environment consisting of various cloud infrastructures by aggregating different IaaS provider capabilities coming from both the commercial and academic area. Recent solutions hide the diversity of multiple clouds and form a unified federation on top of them. Many approaches follow recent trends in cloud application development, and offer federation capabilities at the platform level, thus creating Platform-as-a-Service solutions (eg. Heroku, CloudFoundry, Apcera Continuum). In his chapter we plan to investigate capabilities of these tools: what levels of developer experience they offer, how they follow recent trends in cloud application development, what types of APIs, developer tools they support and what web GUIs they provide. Developer experience is measured by creating and executing sample applications with these PaaS tools.

Keywords: Cloud Computing, PaaS, Cloud Federation, Application development, Developer Experience, Cloud Deployment, Provider Capabilities, PaaS Classification

INTRODUCTION

Cloud computing providers offer services according to several models which can be categorized as follows: (i) Infrastructure as a Service (IaaS) – this is the most basic cloud-service model, where infrastructure providers manage and offer computers (physical or virtual) and other resources, such as a hypervisor that runs virtual machines. IaaS clouds often contain additional resources such as a virtual machine disk-image marketplace with pre-installed images, raw block storage and other file or object storage, load balancing, IP addresses and VLANs. Providers supply these resources on-demand from their large pools of computers installed in their datacenters; (ii) Platform as a Service (PaaS) – these providers deliver a computing platform, including an operating system, programming language execution environment, database and web server. Developers can develop and run their software applications on the cloud platform without the cost and/or complexity of buying and managing their own server farms to match application demand. Software as a Service (SaaS) – such providers install and operate application software in the cloud and cloud users access the software from cloud clients. Cloud users do not manage the infrastructure nor the platform on which their application runs. Load balancers distribute the workload over the set of allocated virtual machines. The load balancing is transparent to the end user, who only sees one entry point to the application. This model has the potential to reduce IT operational costs by outsourcing maintenance of hardware and software to a provider, enabling the reallocation of IT operation costs to other goals.

According to a recent study, the Dzone Guide to cloud development (Dzone, 2015), over half of the companies use Cloud computing in their development (53%), testing and quality assurance (44%), production and deployment work (52%), and these percentages are going up around 10% in surveys from the past years. Lot of the survey respondents replied that they are “planning to perform” testing, development and deployment in the cloud. This shows that cloud is growing ever more important in our

world. From the over 600 IT professionals that responded to the survey, 50% see hybrid cloud as their ideal platform, and private cloud is second with 29%. When asked about hosting types, respondents preferred third party (56%) over on premise (41%). A good representation of the market could be seen by the data gathered in the survey. Respondents of the survey are most likely to deploy web applications (73%) and enterprise applications (54%) in the cloud. They are most likely to use Paas and IaaS types of service. It is interesting to note that Storage-as-a-Service and Database-as-a-Service have risen, and this relates to the impact of Big Data technologies in cloud environments.

Cloud computing is already a part of our everyday life, and there is so much data produced by humans and their machines. As the technology evolves, new kind of innovative usages can be invented. These new capabilities are the motivation for developing even better cloud infrastructures (Rajkumar Buyyaa, 2009), (Michael Armbrust, 2009).

There are some concepts to create one more abstraction layer above infrastructure cloud providers, such as (David Cunha, PaaS Manager: A Platform-as-a-Service Aggregation, 2014). Some of these offer standard APIs (David Cunha, A Platform-as-a-Service API Aggregator, 2013), while others try to avoid vendor lock in situations (Kolb S., 2014) (Sellami M., 2013). These approaches are important in creating a standard cloud platforms, but it is hard to integrate innovative ideas and their implementations.

One of the most critical part in cloud computing in general is security issues. The cloud technology is relativly new, and it has to gain the trust of its users (Yanpei Chen, 2010). The nature of it makes it hard, because in many cases the user doesn't know where or how the data is stored. The cloud is used by many users at the same time and the providers responsibility to make sure that the applications can't affect each other without permission (Luis Rodero-Merinoa, 2012).

The aim of this chapter is present an overview of the state-of-the-art Platform-as-a-Service solutions that are used to develop applications over clouds. We investigate and give an overview of the capabilities of the most advanced providers: what levels of developer experience they offer, how they follow recent trends in cloud application development, what types of APIs, developer tools they support and what web GUIs they provide. Primary sources for this investigation are public documentation of the relevant tools, research publications and trial or demo versions where applicable. Developer experience has been measured by creating and executing sample applications with some of these PaaS tools.

There are many PaaS solutions nowadays, therefore we tried to select the most popular ones that are used by many developers and has stood the test of time. But we wanted to include some relatively new providers because of their innovative or specialized features. Some solutions were sorted out because of the instability and unacceptable quality.

The main challenge we identified during our research is that it is difficult to objectively compare providers belonging to the same category, as they are offering or providing not exactly the same set of services. Providers can advertise such qualities that may not be available or measurable compared to other providers. Therefore we followed the idea that there may be a difference between the feature that the user or developer experiences, and the features the providers advertise. Comparison data may be subjective and biased because of this, but we tried to remain as objective as possible. Similar research efforts have been performed in previous years by others (Pivotal, 2014), (Rimal, Choi, & Lumb, 2009), (Hoefer & Karagiannis, 2010).

The structure of the chapter is as follows: in Section 2 we give an overview of general PaaS providers and present a classification of their main properties both in theory and in practice. Section 3 introduces and compares providers specialized in mobile support. Section 4 summarizes our findings with a discussion, and Section 5 concludes the chapter.

CHARACTERISING GENERAL PAAS PROVIDERS

Overview of the investigated providers

Heroku

One of the first cloud platforms, Heroku (Heroku, 2015) has been in development since 2007, starting with support for Ruby, and adding support for many languages through the years, such as Java, Node.js,

Scala, Clojure, Python, PHP and Perl. Heroku was acquired by Salesforce.com in 2010, as a subsidiary. Heroku's services run on the Amazon cloud systems. From the Developer Experience point-of-view, Heroku's interface is well-polished, intuitive and easy to use. Many times Heroku has been seen as an example by other PaaS providers, for their ease of use, features and reliability. For example Deis uses a Heroku inspired buildpack system in their deployments. The basic units of computing power in the Heroku ecosystem are the Dynos. A Dyno is a lightweight, isolated container that runs an instance of the application.

CloudFoundry

CloudFoundry (CloudFoundry, 2015) is an open source PaaS service, originally developed by VMware, later owned by Pivotal Software, primarily written in Ruby and Go. CloudFoundry is available in three flavors, Cloud Foundry OSS, an open source project available to everybody, which uses the developers own infrastructure and the BOSH shell to interact with it, Pivotal Cloud Foundry, a commercial product from Pivotal, which includes extra tools for installation and administration, and Pivotal Web services, which is an instance of Pivotal Cloud Foundry hosted on Amazon Web Services. Applications deployed to CloudFoundry access external resources via Services. All external dependencies such as databases, messaging systems, file system, etc. are Services. When releasing an application, the developer must specify the Services it should use. Many pre-defined services are available via an administration console, such as MySQL, PostgreSQL, MongoDB, etc. as database services, RabbitMQ as a messaging service, Jenkins for continuous integration, and API Gateway, Data Sync, Push Notifications for mobile development.

Apcera Continuum

Apcera (Apcera Continuum, 2015) is a startup company, launched in 2012, and was founded by some of the architects of CloudFoundry. Apcera Continuum lets companies create, run, manage applications in ways that conform to security and governance policies. Continuum works with a cloud based on on-premise servers and hybrid environments. It's aiming to provide a way to build and deploy applications with IT policies factored in from the start, and to apply policies and governance to existing applications as well. Continuum blends the models of IaaS, PaaS, SaaS but overlays them all with technology that handles policy. Single, Apcera's vision of the future is an operating system for hybrid cloud, which they call Hybrid Cloud Operating System. Hybrid Cloud OS is in development at the time of writing.

Google App Engine

Google App Engine (Google App Engine, 2015) is a cloud computing platform for developing and hosting web applications. Applications that are run in the system are sandboxed, and run across multiple Google-managed servers. App Engine offers automatic scaling for web applications, automatically allocates more resources for the application to handle additional demand. App Engine supports a wide variety of programming languages such as: Python, Java (and by extension Groovy, JRuby, Scala, Clojure), Go, PHP, and many web frameworks. App Engine provides infrastructure to make it easy to write scalable applications, but can only run a limited range of applications designed for their infrastructure. App Engine requires developers to use only its supported languages, APIs and frameworks. By default data is stored in a BigTable non-relational database, and applications that require a relational database will not run on App Engine without modifications such as Google Cloud SQL. Some developers expressed worries that their applications will not be portable from App Engine and fear being locked into the technology. To address these concerns, a number of open-source projects have appeared to create backends for the various proprietary APIs of App Engine, such as AppScale.

Microsoft Azure

Azure (Microsoft Azure, 2015) is a Cloud computing platform, which allows developers to publish web applications running on different frameworks, written in different programming languages such as any .NET language, node.js, php, Python and Java. Azure Web Sites supports a website creation wizard that can be used to create a blank site or use one of the several pre-configured sites. Developers can add or modify content of the website via multiple deployment methods: TFS, FTP, CodePlex, GitHub, Dropbox, Bitbucket, Mercurial or git. Developers can select the place where their website will be hosted from several Microsoft data centers around the globe. Azure Traffic Manager routes traffic manually or

automatically between websites in different regions. Web sites are hosted on IIS 8.0, running on a custom version of Windows Server 2012.

Amazon Web Services

Amazon Web Services (Amazon Web Services, 2015) is a collection of services that make up a cloud computing platform, which are based on 11 geographical regions across the world. The most central and well-known services are Amazon EC2 (Elastic Compute Cloud) and Amazon S3 (Simple Storage Service). The products are offered to large and small companies as a service to provide large computing capacity faster and cheaper than the client company building and maintaining an actual physical server farm. The simplest way to get an application up and running on AWS. The service automatically handles the details such as resource provisioning, load balancing, scaling and monitoring. On can create applications in PHP, Java, Python, Ruby, node.js, .NET, Go or in a Docker container that runs on an application server with a database. An environment using the default settings will run a single Amazon EC2 micro instance and an Elastic Load Balancer. Additional instances will be added if needed, to handle any peaks in workload or traffic. Each Amazon EC2 instance is build from an Amazon Machine Image which can be an Amazon Linux AMI or an Amazon Windows Server 2008 R2 AMI by default.

Mendix

Mendix (Mendix, 2015) is a Netherlands based enterprise application platform company, which has developed a PaaS product called the Mendix App Platform. The platform allows the users and developers to build, integrate and deploy web and mobile applications. The Mendix App Platform uses a visual, model-driven software development approach. Mendix is also a member of the Cloud Foundry Foundation. The company plans to evolve the open source standard as well as integrate the PaaS framework of Cloud Foundry with its own rapid app platform as a service. The rapid app platform as a service methodology prioritizes speed and agility and utilizes a "no code" principle. Using visual models simplifies app creation and migration and allows business users more insight and more direct participation in the development process. This method reduces the time to market of the app significantly.

Services and applications in the Mendix App Platform are oriented towards collaboration. Mendix offers a Community Edition of the app platform, which bring application development tools to a wider audience ranging from individual developers to large teams and companies. Recently Mendix made major enhancements to the Mendix App Platform, such as reducing time, cost and effort for app development and mobile application development. Mobile features include pre-defined layouts for smartphone, tablet and desktop user interfaces.

CloudControl

CloudControl (CloudControl, 2015) is a programming language agnostic PaaS service based in Germany. Its officially supported development languages are Java, PHP, Python and Ruby, and the services use the open buildpack API developed originally by Heroku. Pricing is based on usage. Computing containers are billed in Boxes per hour, where one Box is similar in power to a Dyno with Heroku, with the notable difference that it can answer two requests simultaneously. CloudControl features deployments that enable separate versions of the application (e.g. development, staging, testing, live). A role based permission system is provided to authenticate multiple developers to work on the same application.

AppScale

AppScale (AppScale, 2015) began as a research project at the University of California. It is an open source computing platform that deploys and scales unmodified Google App Engine applications over public and private cloud systems and on-premise clusters or a hybrid of these systems. The service is modeled on the App Engine APIs and supports Python, Go, PHP and Java languages. They are in close partnership with Google. One of the goals of AppScale is to make the applications of the developers truly mobile, to provide a way to migrate or fail over apps to another cloud or the developer's own servers if needed. It minimizes the problem of vendor lock-in. They aim to provide developers with a rapid API-driven development platform that can run applications on every cloud infrastructure. AppScale uses many open source components and APIs such as Cassandra for data and blob storage,

memcached, RabbitMQ for queueing, ejabberd for messaging, etc. AppScale can be run on the Google App Engine and in the Amazon Web Services cloud.

Deis

Deis (Deis, 2015) is a lightweight application platform that deploys and scales Twelve-Factor apps as Docker containers across a cluster of CoreOS machines. The Twelve-Factor app is a methodology for building modern applications that can be scaled across a distributed system. It provides a lightweight PaaS with a Heroku-inspired workflow (Twelve-Factor app, 2015). Deis creates the applications as Docker images, which are distributed across the cluster as Docker containers. It uses CoreOS, a minimal linux distribution that can be run and hosted everywhere, public or private, bare metal or virtualized. It uses Git as the main version control system, the developers can deploy new content with a simple "git push". It is free and open source.

BlueMix

IBM's latest cloud software is the BlueMix platform (Bluemix, 2015), which is using the CloudFoundry opensource PaaS project. It's an implementation of the IBM's Open Cloud Architecture. Bluemix delivers enterprise-level services that can easily integrate with your cloud applications without you needing to know how to install or configure them. For developers, Bluemix further optimizes the time you spend creating cloud application. For organizations, it provides a cloud platform that requires very little in-house technical know-how as well as cost savings.

PaaS comparison categories

We compared the PaaS providers based on the following categories:

- **Cost** - The cost of the service can be a big factor in choosing a provider. Different tiers of service may be available for different fees. There are many different billing schemes, which differ on what kind of resource consumption they track.
    - Most providers offer free tiers with limited processing power, database size and connections, network bandwidth, and data storage.
    - Pricing is done in a pay-as-you-go system with most providers, by billing them based on the resources used monthly.
    - Higher tier services may increase monthly cost. These include a separate database instance instead of a shared one, High Availability, data rollback, encryption, different (higher) SLAs.
- **Workload** - With this category we tried to define what kind of workload and what type of data works well with each provider. The basic kinds of workloads that we were interested in are:
    - Analytics - This type of workload is about crunching the numbers, analyzing and processing data. This may be done in a distributed way, like Hadoop Clusters.
    - Transactions - With this kind of workload, stress is on the database actions, transactions, and data retrieval.
    - Media - Working with media, delivering large static files and other content.
    - Mobile - Do they offer any mobile apps and a backend?
    - IoT – Internet of Things support
- **Tooling / Developer experience** - This category is about how easy it is for a developer to use the service and develop his own application on a day-to-day basis.
    - IDE support - More and more providers supply developers with plugins and extensions to the major IDEs, (e.g. Eclipse, Visual Studio, NetBeans, IntelliJ, etc.)
    - Targets - Some providers offer the ability to run different instances of the applications simultaneously. One instance might be the live application with actual customer or user data, a second instance can be available for developers to test their new code of the application, maybe a third for testers to run their tests against, etc.

    - APIs - Providers may offer some public API endpoints for the developer to interface their own application with. Different interfaces can be used for parts of the service.
    - Command Line Interface - Most of the providers offer a Command Line Interface for managing the service. These can vary wildly in quality and usability.
    - Git - Nowadays Git is the most used distributed version control system. How (and how well) is git used with the service? Is it used to deploy or push newer versions of the application?
- **Deployment** - Points in this category are about the deployment of the service and the containers used / promoted by the provider.
    - Backend type - The big providers (Google, Microsoft, Amazon) have their own server farms that run the PaaS applications. In contrast, smaller providers host their services on Amazon servers, by using their own PaaS solution on top of the IaaS provided by Amazon.
    - Container solution - Containers are the building blocks of PaaS services. In the age of virtualization, some different container types have appeared. One of the main container technologies is Docker, which many of the providers support. Other providers may use a different kind of method, or have a system based on Docker.
    - Supported containers - If the provider supports more types of containers, that might be beneficial for developers, because they might not need to rewrite their application for a new container solution.
- **Integration** - Some providers may offer solutions for managing the users of the application, (e.g. a registry form, login pages, errors, and user management interfaces for administrators).
    - User registry - Adding, registering, managing, removing users in the application, and managing the details of the users.
    - Authentication / Authorization - Methods, granularity of managing the access of users to content/data.
    - Security - Security of users and user data.
- **SLAs** - Providers agree on some levels of service that is definitely provided by them. This could be imposed on uptime, scaling, performance, some kind of support for paying tiers of service, etc.
    - Availability / HA - The type and amount of availability guaranteed by the provider. High Availability may be available for redundancy, switching over to other servers when one node goes down in the system.
    - Reliability - How many and what size outages happened in the last few years of service?
    - Horizontal scalability - The process of allocating more nodes to serve a growing number of requests on the application.
    - Vertical scalability - The process of adding more resources (more processing power, more memory, more disk) to a node.
    - Horizontal and vertical scalability may also depend on the application itself.
- **Datastores** - If the provider offers any kind of data storage methods, that are mentioned here.
    - Relational (SQL), NoSQL, Key-value, Documents, etc. - The provider may offer different types of data storage, with different backends and APIs.
- **Programming models** - The more different languages and frameworks a provider supports, the more potential customers it might attract.
    - Programming languages - Amount and type of supported programming languages available for developers to use for writing their applications.
    - Frameworks - Providers might support web frameworks, which may be pre-installed or easily installed from some kind of marketplace of the provider.
- **Management** - Managing the events and availability of the service can be done on many levels, they are aggregated in this category.
    - Logging - Different levels and methods exist for logging different types of events or activities. Logs can be accessed in many different ways, there are some standard methods and frameworks (e.g. LogStash, etc.), which provide easy access and different search parameters and functions to find the relevant data.

    - Monitoring - Monitoring the quality of the service and the application is an important aspect of the user and developer experience. Alerts can be sent out if different service-based failure criteria are met, developers and administrators can be notified. Monitoring is usually available as self service monitoring for the free and lower tier users, and enterprise grade monitoring might be available for higher tier customers.
- **Misc**. - Any other criteria that could not be fitted in their own group is mentioned here.
    - Open source - It might be important for a developer to know if the source of any parts of the provider solution is available.
    - Marketplace - More and more providers offer an in-house marketplace of selected, pre-assembled containers or software images, made by the PaaS provider or even other developers.

Features in categories (where applicable) were measured, aggregated and ranked on the following scale:

In our comparison table we used numbers, there are 4 different levels. 0 means the provider has not implemented the feature, or there is no data available. 1 means the feature quality is basic, beta state, can have major bugs. 2 indicates a stable quality. 3 means the feature is excellent or innovative.

		Apcera Continuum	Heroku	CloudFoundry	Google App Engine	Microsoft Azure	Amazon Web Services	Mendix	CloudControl	AppScale	Deis	Bluemix
Cost	Billing schemes	n/a	free, scaling	free	free quota, priced above	free, pay-as-you-use	pay-as-you-go	free, 3 tiers	free, pay-as-you-go	N/A	free	free quota, priced above
Workload	Media, larger files	na	2	2	1	3	3	2	1	1	1	2
	Mobile	na	2	2	1	2	1	2	1	1	0	3
	IoT	0	2	2	3	3	2	0	0	0	0	2
Tools	IDE support	na	2	1	1	2	1	1	2	2	2	1
	Targets	na	3	3	1	3	2	2	3	1	2	3
	APIs	2	3	3	1	1	3	2	2	3	1	3
	Git usage	na	2	3	2	1	2	2	3	2	3	3
	CLI	2	3	3	2	1	3	1	3	2	3	3
Deployment	Backend	private	Amazon	Amazon	Google	Microsoft	Amazon	public, private	Amazon	Google, Amazon	Amazon, own	SoftLayer
	Own container	3	3	2	1	1	2	0	0	2	na	2
	Supported containers	Docker	na	na	na	na	na	na	buildpack	GAE	buildpack, docker	Docker
Integration	User registry	2	1	2	2	3	2	2	2	2	1	2
	Authentication / Authorization	2	2	2	2	2	3	2	2	2	1	2
	Security	3	2	2	2	2	2	1	1	2	1	2
SLAs	Availability - HA	na	2	1	3	3	3	1	1	1	2	2
	Reliability	1	2	3	3	3	3	2	2	2	2	2
	Vertical scalability	na	3	3	1	3	3	2	3	3	3	3
	Horizontal scalability	na	3	3	3	3	3	2	3	3	3	3
Datastores		SQL, K/V	•SQL, •Doc, •Graph, •K/V	•SQL •K/V •Doc	•Column-based, •K/V	• SQL • K/V • Doc • Graph	•SQL •Column •K/V	na	na	3	1	SQL K/V

		Apcera Continuum	Heroku	CloudFoundry	Google App Engine	Microsoft Azure	Amazon Web Services	Mendix	CloudControl	AppScale	Deis	Bluemix
Programming models	Languages	3	3	3	3	3	3	1	2	3	2	3
	Frameworks	2	3	3	1	1	2	1	2	2	1	3
Management	Logging	2	3	3	1	2	2	1	1	2	2	3
	Monitoring	2	3	2	2	2	3	2	2	2	2	2
Misc	Open Source	yes, except core	no	yes	no	no	no	has Community Editon	no	yes	yes	based on open source
	Marketplace	yes	yes	yes	yes	yes	yes	yes	yes	no	no	yes

PaaS Workflow of developer roles

The basic developer roles and their connections in PaaS application development is depicted in Figure 1. In general, after the **Authentication** the user can **Edit**, **Run** or **Monitor** applications. In the authentication part the user can **register** or **login** and then use the features of the given PaaS system depending on the user's role and **permissions**.

After the authentication in the **Edit** part, the user can create a new application in the system or use an existing one. The source code can be edited, the runtime is automatically recognised by the system. The dependencies can be modified by language specific ways, usually Java has maven *pom* files, Ruby use a *Gemfile*, in Python there is a *requirements text file*, while in Node.js a *package.json*, and so on. The execute types and commands can be added to the application, but some defaults are recognised by the system. For example in Ruby on Rails, it's typically rails server, in Django it's python <app>/manage.py runserver, and in Node.js it's the main field in package.json. Some configuration variables can be added, these are good for make different settings for instances running the same code, for example add credential keys. The addons can provide extra functionality and services like logging, performance monitoring, data storage, etc.

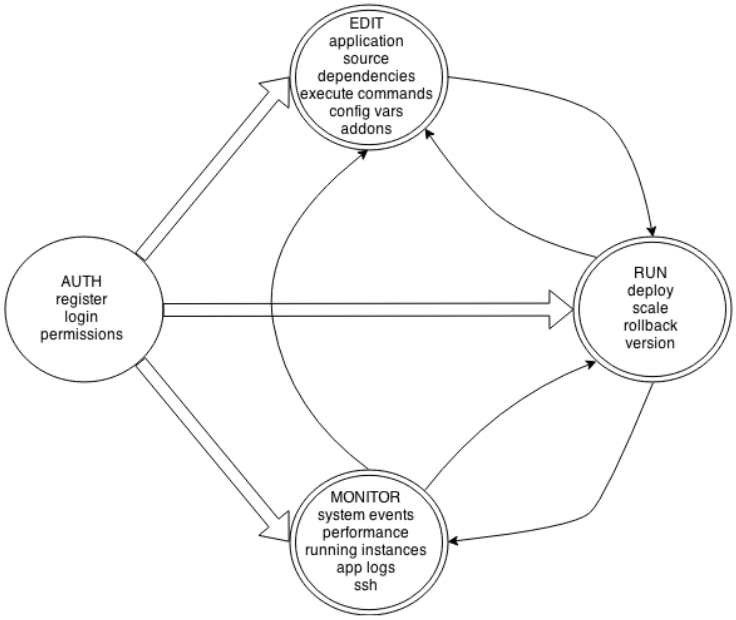

Figure 1. Basic developer roles

The other option for the authenticated user is to **run** the application. The first thing is to deploy the application, this usually made with a VCS or with direct upload, the PaaS system recognises the runtime and the dependencies and builds the application. With the recognised or given execute commands the user can select one and run it on the cloud. With starting more instances the application can be scaled, for example starting more web frontend instances the application can handle more visitors with the help of a load balancer. To handle more workload some worker instances can be started. To provide high availability is done with the same way. If there is an issue with the latest release of the application, the user can use the rollback feature to restore a previous version. It's useful to have more versions of the application, for example a stable release for production, an unstable beta version with new features to demonstrate, and some developer versions for testing.

The **monitoring** part is interesting for managers and testers too, not just for developers. The system events and the application logs can be seen for debugging purposes. The application's performance can be analysed, and for example with the list of the running instances types and numbers the application's scalability can be diagnosed. The ssh and other type of connection with a running instance is useful for debugging or run one-time bugfix scripts. Every PaaS provider has its own tools and methods to provide these general features to the users.

The transitions between the main states (Edit, Run, Monitor) are shown with arrows. These sate changes should be imagined like an iterational application development workflow. For example after implementing a feature the developer deploys the application, and if it fails, based on the logs we can go back to the Edit state, otherwise the Monitor state can be used to analyse the application behaviour, for example the ability to scale. After drawing the conclusions, based on the performance, we can go back again to the Edit state to fix some noticed problems, or change the number of the running instances for further investigation or testing.

We took these identified roles into account for our practical investigation of the most widespread PaaS providers by developing and running a sample application for all of them. We introduce our experiences in the next subsection.

Experiences of PaaS application development with the main providers

In order to study PaaS provider's developer experience from the practical side, we decided to create a simple web application and deploy it to different platforms to see what the biggest providers offer nowadays. We addressed the following providers: **Azure** (Microsoft Azure, 2015), **Heroku** (Heroku, 2015), **OpenShift** (OpenShift, 2015) and **Continuum** (Apcera Continuum, 2015). We also spent some considerable time with **Cloud Foundry** (CloudFoundry, 2015), but it was so instabile that we couldn't get any proper work done with it. We desided to use BlueMix (Bluemix, 2015) which is strongly based on Cloud Foundry. There were some hindering issues at the beginning, but with each modification that had to be done in order to get an application working at a certain provider, the application became better, more resilient and, in many cases, more convoluted.

We decided to create our application in **Python** (Python, 2015) and **Node.js** (Node.js, 2015), therefore two versions of the same album application were made. While it is not exactly meant for real life use, it aims to make use of various PaaS functionalities. During our tests some of the provider's services have changed, e.g. Azure ruining its interface even harder, requiring a credit card number even for the free services, and Heroku imposing an obligatory 6 hour sleep on free accounts.

Our test application is a fairly basic image hosting service. Its components are:

- **Website**: for managing image uploads, comments and albums. The Node version also utilises websockets.
- **Worker**: for resizing images and uploading them to a storage.

Our main aim was to separate the website and the worker instances as much as possible, making them independently scalable, but still connected, either through sharing the same code and services. The application has the following requirements:

- A **Relational database** for storing image, user and album data. While MongoDB (MongoDB, 2015) is more popular Node users we still opted for RDBMS's in order to share the database between both Python and Node versions. We're using ORM's in both implementations, for both comfort and practical reasons.
- A **Redis server** for acting as a basic message queue, a session store and sharing the image processing status through basic keys and/or pubsub.
- A **MongoDB** server was needed when using the Celery version of the Python worker because it consumed most Redis providers' connection quota.
- Some kind of **blobstore** to store the images themselves. Current supported backends are Azure, Cloudinary (Cloudinary, 2015) (both with their own SDKs) and local file system, the latter can be used for NFS too.

In both implementations the workers are using the **ImageMagick** (ImageMagick, 2015) library in one way or another, which gave some complications. When we ran into issues with either of these aforementioned services given by the PaaS providers themselves, we used some third party hosts who offer services even outside of PaaS providers' reach, like:

- **MongoLab** for MongoDb,
- **Redis4You** (Redis4You, 2015) for Redis,

- **ClearDB** (ClearDB, 2015) for MySQL, and
- **Cloudinary** for blob storage (which is an odd one out since it features its own API).

This approach was most necessary on Azure, when all MSSQL libraries on both platforms were unusable and registering a Redis and any other database required a credit card. When everything else failed or we felt were very slow, we hosted them on localhost for local testing or in an Azure based Linux VM.

Concerning general deployment isseus, every popular web based PaaS provider offers git support. The way they are handling it is more or less uniform, such as following the deployment messages during the push, being able to revert previous revisions through the web interface and such. The biggest difference between them is the way of authentication, but this is really just a minor detail. Azure requires authentication through git itself, while OpenShift and Heroku have their own tools. A minor issue we had was that most of their git repos weren't empty when creating new apps, but contained a sample application. Whenever we started a new app we had to create and empty local repo, add the one given by the PaaS provider as a remote, delete everything, add the remote to our project's local repo, pull it again and finally push it to deploy it. Azure can give an empty repo, so one can skip this drudgery. Sharing the same repository was perfectly doable, the differences of the different platforms were only had to be taken into account during the credential setup stage.

As mentioned before, some cases we had to use third party services. A lot of PaaS providers have marketplaces, where users can bind services to their applications with different pricing packages that are included in the user's subscription. Most of the time the services work as expected, without any major restrictions or special connection parameters. There are of course some exceptions. A lot of Redis providers require to have the "ready check" parameter disabled. A lot of the require passwords, which cannot be passed through connections strings with some bindings, such as Node's node-redis (node-redis, 2015). Azure's Microsoft hosted Redis service requires SSL by default, which can be turned off. A lot of these only can be accessed from the PaaS providers network, not from outside, such as the developer's system. While this is perfectly understandable, for some operations the using the REPL from the app's shell isn't always feasible. This came apparent to us while we were trying to get rid of same MySQL tables but because of the foreign key constraints we couldn't. A SET foreign_key_checks=0; (MySQL table deletion workaround, 2015) instruction could've spared us a lot of headaches. Azure gets around this issue with being able to add single IP exceptions to its database firewall and OpenShift by supplying a phpMyAdmin (phpMyAdmin, 2015) interface. Adding command line clients to the app shell could be also a nice solution.

Regarding language-specific requirements, in the Python 3 version a lot of useful libraries were not ported yet, such as the majority of the Memcache (Memcache, 2015) ones for caching, gevent (gevent, 2015) for setting up a lightweight server to get around health checks, protobuf (Protocol Buffer, 2015) for precise data formating or supervisord (Supervisord, 2015) for process management. Therefore we had the following additional requirements:

- **RDBMS**: it's handled through the SqlAlchemy ORM, due to most service providers shipping with different database services by default (Continuum and Heroku with PostgreSQL (PostgreSQL, 2015), Azure with MSSQL, 3rd party free MySQL hosts) and it also has an amazing migration framework.
- A **Redis** server for communication between the worker and the website. It can also double as the message queue when the connection count constraints allow it.
- A **MongoDB** indirectly through PyMongo (PyMongo, 2015), because only Celery is using it.
- A **storage backend** to store images, since most service providers don't have any support for permanent file storage. It currently supports
    - **local storage**: as basic file operations
    - **Azure blobstore**: Microsoft's own SDK worked fine
    - **Cloudinary**: their own SDK (Cloudinary SDK for Python, 2015) worked fine

For image processing we originally used the **Pillow** (Pillow, 2015) library, as used by many Python developers, however installing it was not really user friendly, so we switched to **Wand** (Wand, 2015), an ImageMagick based solution. In the end we called convert (convert, 2015)directly on Continuum.

Concerning the Node.js version, creating the application was way less difficult than the Python one. The basic idea of communicating through environment variables of these modern PaaS services stayed the same, aside from them variable names themselves other thing didn't have to be changed. The application structure stayed the same, the following libraries were used:

- **Website**: We opted for Express 4 (Express, 2015) as every other Node developer. Its session store makes use of Redis as a backend. It notifies the client of the image processing status in realtime through websockets using the Socket.io (Socket.io, 2015) library.
- **Worker**: The same BLPOP queue applies here too. Invoked as an external process from the website through forever. It also starts its own HTTP server through the default http (Node.js, 2015) module to fool the health check.
- Requirements:
- **Database**: Uses Sequelize as the ORM for the same reasons, as it was already behind an Sqlite, MySQL and PostgreSQL database.
- **Redis**: Had some issues with the authentication since most hosts require passwords and by default our localhost Redis instance didn't. Passing the password in callbacks isn't a beautiful sight using the "redis" library. In addition, the Redis connection's ready check setting had to be disabled on external services, which we wouldn't have guessed by ourselves.
- **Storage**: We used easyimage to resize the images, which is an ImageMagick backed solution. For uploading, Cloudinary's and Azure's API worked as expected.

In the following we detail our experiences with the concerned providers.

OpenShift

In a nutshell during Node deployment, it did what it promised, we didn't have any issues with it. With Python however we ran into a major issue in the early stages of deployment, namely it refused to install packages from requirements.txt, which was a major stumbling block. It seems like a bug, but since it hasn't been fixed on such a large system from a long time makes us this otherwise. The SSO links on the dashboard to the 3rd party services rarely ever worked. Fortunately every information we needed was in environment variables. What we really like was that there was nano on its SSH shell. Scalability is not exactly streamlined, one has to choose whether he wants to scale its app or not when creating it, conversion between these two types isn't possible.

Heroku

Its application-scaling was very close to how we imagined separating the worker and the web application. They use the same repository, they are independently scalable, and share the same 3rd party services. Initial start-up commands are taken from **Foreman**'s Procfile (Foreman and Procfile, 2015), to save an extra step after deployment. It recently also started its own Redis service (Heroku hosted Redis, 2015), which offers bigger storage that any other competitors for free of charge (20MB vs 10MB).

Azure

Since applications in this platform use Windows operating systems, some major modifications were necessary on some parts of our application. The administration website was generally slow. It provides a web based Powershell shell, but on the only occasion we felt it was necessary to use, it failed to work: we wanted to download a somewhat big archive to extract some DLL's from it, but its HTTP download command refused to run because Internet Explorer wasn't configured yet. It provides a Web Job (Azure Web Job, 2015) abstraction, which allows setting up recurring tasks as: (i) one shot, (ii) scheduled and (iii) continuously running tasks, which was somewhat what we were looking for while designing the worker setup. While we managed to get it working, it required a lot of work and a lot of language specific workarounds. To be specific, sharing libraries between the web app and the Web Job is a hard task.

It's rich with self hosted and third party services. We only used the blobstore through the Azure SDK (Azure SDK for Python, 2015), (Azure SDK for Node, 2015), because the message queue had a really small message size limit, making it unfit for the application. The MSSQL server it provides would've been also a good candidate for testing, but none of the ORM's we used (Sequelize (Sequelize, 2015) and SQLAlchemy (SQLAlchemy, 2015)) were supporting it at all. With the previous three sharing services

between different apps weren't necessary, resulting in multiple registrations on the same email address to the same service.

Continuum

The way it can share and bind services between applications turned out to be really useful, while the distinction between services and providers is not really trivial at first. While it's a matured and well documented system, We've met some bizarre behaviors while checking it out. The handling of NFS volumes can be tricky. We don't really understand what has been happening, but occasionally some mounted volumes work, sometimes don't and sometimes they even terminate apps. This approach sometimes worked, sometimes it did not. When it did not work, we experience either of the following: The app refuses to start, but ssh-ing into it shows, that the volume has been mounted, or not event ssh-ing works anymore.

The app never starts again, even after unbinding and deleting the NFS services and providers. The way that we could get it to work again was to delete the application and recreate it. There weren't any proper error messages and the logs yielded nothing. We had to start a dummy HTTP server on the worker processes in order to fool the health check, since it was communicating only through Redis with the outside world and wasn't serving a website. The lack of text editors really take away from the SSH's usefulness, resulting in regular redeploys in the application's infant state, which of course in lengthened by the fact, that staging redownloads every single dependency during each deployment.

Cloud Foundry

We tried installing Cloud Foundry through the bosh-lite (Bosh-lite, 2015) based method, but it failed most of the times with a different error every week. These errors ranged from missing deployment manifest files, malconfigured default nginx configurations to dying connections during blob downloads. We eventually gave it up when VM snapshots couldn't be restored without the virtual network cards dying. There were short periods when it was working, but after additional efforts to make it more usable, such as installing 3rd party service packs for MySQL and Redis, it refused to work anymore and even restoring snapshots couldn't save it.

IBM BlueMix

Persistence had to be handles with special care, because it does not provide any blobstore service so far. Our app uses the following two solutions:

- MongoDB's GridFS feature to store files alongside the database, and
- Cloudinary, a 3rd party service reachable through an API using a Node.js package that specializes in media storage

The only Bluemix (or rather Cloud Foundry) specific step was to read the application's host and port from an environment variable, and the application was ready to start. Imagemagick was installed on the app's container by default, so we cloud use it easily. Websockets were also supported outside-the-box.

IBM offers a marketplace for self-hosted and 3rd party services on its marketplace, just like many other PaaS providers. There are ones hosted by IBM itself, including in beta services such as message queues. Since IBM has interest in a company called Compose.io, their services are also plugged into the catalog, including free ones. To access these, however, one must provide his credit card number. And thirdly, there are few third party providers.

For the original application we needed a Redis server, an SQL server and optionally a MongoDB server. We selected Redis since it could be replaceable with MongoDB. We decided to do so, when we've seen that the Compose.io provided Redis servers required additional service registration.

We also needed an SQL database. IBM provided one without any sorts of obligations: a DB2 service. Since there were no drivers for Sequelize, we did use it. Aside from the Compose.io services, there were also some popular third party ones: ClearDB for MySQL and ElephantSQL for Postgres. We used another ClearDB account from another test, because at this point we have already had about 5 of those

tests from other providers. We experienced the same with MongoDB; the only service in the Catalog was also from Compose.io, so we had to use a previously created MongoLab account.

Docker containers created within Bluemix can also be bound as services, which are also visible through environment variables.

The deployment process went really smoothly. The only prerequisite it required was CloudFoundry's cf command line tool. One can deploy either through this tool or through a git repository. To test the deployment, we have chosen the command line method, which only required running a single command the the application's directory after the necessary authentication.

No real special preparations were needed for deployment aside from specifying the required Node version in package.json, which installed the newest version without a hitch. During our test we uploaded the application without the Node.s package folder. Those were also downloaded during installation and cached for further deployments.

The Dashboard provides the essential functions for managing an application, such as editing its description, managing the instance numbers, allocating resources, binding services through the Catalog, managing routes, settings environment variables and browsing logs in real time.

A feature we missed however was settings in its start command. This has to be done either though the (optional) Cloud Foundy manifest file, through a Procfile (only the web field is taken into account), or by the application's running environment (eg. in Node.js's case, the npm run command in package.json)

Bluemix's documentation clearly states that they do not support Heroku's dyno system, meaning that there cannot be two different processes running from the same codebase in different containers. Instead, they suggest uploading the same application into a duplicate app dedicated to the task.

Health checks are done frequently by checking if the application's accepting connections. This means that an app has to have a port open, even if it does not communicate directly.

By the time we evaluated Bluemix, strating an ssh connection into an app's container did not work from command line. On the other hand, they have a web based file browser on the dashboard, however this only supports viewing files.

Docker image can be deployed and bound to apps. To manage these we need to install the appropriate cf plugin, which assumes that Docker is also installed on the developer's machine.

Developer experience comparisons

Table 1 shows a comparison of the PaaS providers we encountered based on the differences we found between them based on the experiences of both platforms.

We used the following expressions in the table:

- **Shareable service**: In some cases binding the same service to different applications can be useful. A method that doesn't involve sharing the connection string directly by hand can make deployment somewhat easier in special cases.
- **Websocket support**: A lot of web applications rely on real-time communication through websockets.
- **Cached packages**: Most packages stash already installed packages away when redeploying the application, making the process way faster.
- **Multiple independent instances of the same app**: For the lack of better wording. Giving some sort of abstraction to run the same code on different environments independently from each other can make the maintenance easier.
- **ImageMagick support by default**: Most Node image processing libs rely on ImageMagick.
- **Text editor in shell & Text editor outside of Git**: Really useful when thing go unexpectedly awry during deployment.
- **3rd party service marketplace**: From what we've seen from the documentation Continuum already has the API and the backend for 3rd party support

- **Tailable logs**: Watching the running instance's output real-time.
- **Git revertable from web**: Following the deployment status and reverting mistakes in cases when using Git isn't an option.

Table 1. Comparison table of PaaS functionalities based on application development experiences

	Continuum	**Heroku**	**OpenShift**	**Azure**	**BlueMix**	**Comment**
Shareable services	+	-	-	-	+	Technically they can be shared by sharing the connection string
Websocket Support	+	+	+	+	+	-
Cached packages	-	+	+	+	+	-
Multiple independent instances of the same app	-	+ (Foreman)	-	+ (Web Job)	-	-
ImageMagick by default	-	+	+	-	+	-
Text editor in shell	-	-	+	-	-	-
Text editor outside of Git	-	-	+ (nano through SSH)	+ (web)	- (read-only)	-
3rd party service marketplace	-	+	+	+	+	-
Tailable logs	+	+	+	+	+	-
Git revertable from web	-	+	-	+	+	-

CHARACTERISING PAAS PROVIDERS WITH MOBILE SUPPORT

MBaaS (MBaaS wikipedia, 2015) (Mobile Backend as a Service) or also known as BaaS (Backend as a Service) is a cloud category. Pre-built cloud hosted components helps the mobile application and web developers to easily have features like data storing, user management, push notification and connection with social media networks. The applications and the backend can be linked with SDKs (Software Development Kits) and APIs (Application Programming Interface).

There are many MBaaS providers in the market with different type of MBaaSs. Two big category is the open source solutions and a non-open source backends. The targeted applications can be different too. There are providers to help startups, simple application developers, game developers with fast and easy development and deployment and the capability to scale. There are solutions for enterprises where it is important the integration capability with other systems and the security.

The list of the examined solutions in this paper are not complete, there were some projects with not acceptable quality that are sorted out. We considered the popularity of the provider in the selection process, but the list contains some new, but innovative solutions too, in these cases the beta state is not a reason to sort out, the vision of the product is remarkable.

Overviewed MBaaS providers

Google

The Google MBaaS's (Google Mobile Cloud Platform, 2015) biggest advantage is that many cloud services can be used, the server or cloud side is really great. There are some specialized services, for example the image processing service, and some general propose ones like Compute Engine. The storage part is advanced too, plus the big data processing is not a problem. The responsible service for the cloud and client connection is the Google Endpoints (Google Endpoints, 2015), which is just an extension to generate client side SDKs. The SDK is ok, because it can handle custom cloud calls, but requires too much coding.

Parse

Parse (Parse, 2015) is really great in the SDKs, it has many SDKs and they are efficient ones. The Android SDK is not just a REST call wrapper, it helps the developers with extra features like push notification for devices without Google Cloud Messaging (Kindle) and there is a library for UI elements to help the login implementation. It has a feature to save the data the next possible time when there is connection. The SDK gives sync and async options for every method. It supports mobiles (iOS, Android, Windows Phone, Xamarin, Unity), desktop and web apps (OS X, Windows, JavaScript, PHP) and embedded devices (Arduino, embedded C). The documentation is examplary and the admin site is really simple and useful with many features. The weakness is the lack of integration with enterprise databases and applications, this is the reason why it's not a good choice for business application developers. The CLI deployment method is not user-friendly. All of the SDKs are opensource projects. Parse support the Internet of Things, meaning different sensors can send data to the cloud and can recive push notifications. Beside the general embed C SDK there are some manufacturer specific SDKs integrated (like Intel).

Backendless

Backendless (Backendless, 2015) has an advantage of letting it host on own or hybrid cloud systems. The most special feature is the media streaming, but this is just for iOS platform. There are some scenarios for working with live or recorded media for example: live video/audio broadcast, recording video/audio content on the server, video/audio chat, video/audio playback for live and on-demand content. Versioning is another great feature, this way the developers can work on a development version next to a stable release.

Amazon

Amazon (Amazon Mobile, 2015) is a PaaS provider, and they have many "blocks" to build applications with. This lets more general usage, but not so many details, that could make the developer's job easier. With the 3 main components (Cognito (User management), Mobile Analytics and Simple Notification Service) the mobile solution is a good piece for the whole Amazon cloud offering. Still not for enterprise, because the lack of integration and security.

Firebase

Firebase (Firebase, 2015) offers a secure, real-time, cloud-hosted, NoSQL database with a login service. This solutions is specialized for low latency. Many features missing, some basics too like push notification and integration, but surprisingly great applications can be written with it, the sample apps greatly demonstrates it.

FeedHenry

FeedHenry (FeedHenry, 2015) is a Mobile Application Platform for enterprises, with many features. Supports the agile development, integrates with git, has many connectors like Salesforce, SAP and Oracle. Concentrates on security. Helps the reusability, has many plugins, templates and a drag and drop application builder, where the mobile application can be developed without coding.

Appcelerator

Appcelerator (Appcelerator, 2015) started as a mobile development tool, to create mobile applications for all platforms with common code written in JavaScript. This main feature has been extended with its own IDE, testing tools and MBaaS features. The MBaaS part is really good for enterprises, it has many connectors, can be deployed with hybrid or fully private clouds. It's a full mobile development platform.

BaaSBox

BaaSBox (BaasBox, 2015) is in beta stage, but will be a stable release in short time. It has many features, they care about the performance too. Promising open source project with Apache 2 license.

DreamFactory

DreamFactory (DreamFactory, 2015) has many install guides for IaaS providers (Docker, Amazon Web Services, Microsoft Azure, Google Cloud Platform, VMware Marketplace, Bitnami Cloud Hosting), for PaaS providers (Red Hat OpenShift, Pivotal Web Services, IBM Bluemix, Heroku) for Desktop Computer (LAMP or WAMP) (Linux, OS X, Windows) IBM SoftLayer, Rackspace Marketplace. For push notification it uses the Amazon SNS. The SDK is not so good, too general, requires too much coding. On the free hosted version the login window pops up for almost every click.

Strongloop - LoopBack

The top features are the modularity, enterprise connectivity, API Explorer, generators, client SDKs. It has some not basic features like geolocation search. StrongLoop Arc is a great tool to visually edit, deploy, and monitor LoopBack (Loopback, 2015) apps.

UserGrid

Apache Usergrid (UserGrid, 2015) is currently undergoing Incubation at the Apache Software Foundation. It already has many basic and not so basic MBaaS features, and it has a potential to be a great solution.

Helios

The Heroku MBaaS solution Helios (Helios, 2015) is in beta stage. The main disadvantage is that the only supported client platform is the iOS. But the the support of this only platform is great, it has some not basic features like In-App Purchases, Passbook, Newsstand, Logging and Analytics, A/B testing.

We also compared the overviewed PaaS providers with MBaaS support. In our comparison table, if a property is measured by a number, the number can have 3 different levels. 0 means the provider has not implemented this feature or doesn't work. 1 means the feature is usable and it's on the same level as the average. 2 means the provider implemented it in an outstanding way.

Comparison of MBaaS providers

We compared the PaaS providers based on the following categories:

- **Price:** This category isn't really detailed, just indicates that what kind of prices have the product. It can be time or function limited free trial, pay/month or pay/use. But it can be important, because it's a good offer for startups, but if the application has more and more users and the bandwidth is growing the bill can be extremely big.
- **Open source:** The open source has its benefits like security, quality, customizability, etc. If it's open source, the license type is shown too.
- **Hosting:** Some backend can be hosted on optional clouds, even on private cloud too, some others are strictly hosted by the provider, and others can have both options. The hosting can

have more categories, like multi tenant or dedicated, and if a private cloud hosts the application, the deployment can be managed by the provider or it can be the developer's job.

- **Custom business logic:** Can a developer customise the server side code? The type of the customisability can be different, it can be limited to add triggers on data modification or highly customisable when the developers can add their own api methods too.
- **Server side language:** If the server side can be customised, it can be done in one or more languages. The support of more languages is preferred because the developers can use the language that is closer to their knowledge and fits more to the task. The reusability is an options here too.
- **Admin site:** The admin sites main function to manage the applications. The main goal is to be simple and still usable with many features. The admin site helps the developer or the manager to browse the data, send push notifications, see the analytic results and many more.
- **SDKs:** The number of supported platforms is important, because usually an application wants to reach as many people as it's possible with minimal effort. The most common platforms are the Android, iOS and JavaScript, but some providers has many more. The quality is more important than the quantity, it's important to be efficient with a few lines of code and still have the ability to handle custom actions.
- **IoT:** The Internet of Things capability is a new trend, many primitive sensor can provide data, if the cloud can gether the information it can analyse it and visualise it.
- **Documentation:** The documentations evaluated by their understability, accuracy, currency. Usually the documentation is an online website with fresh informations about the platform.
- **Tutorials:** The tutorials evaluated by their coverage, quality and variety. To start using the platform, it's great to have clear step-by-step instructions, and visions for other possibilities.
- **Sample apps:** The tutorials evaluated by their realisticness, quality and variety. It's a great help for developers to see a working example with no effort, to see the complexity and usability of the provider's solution.
- **Storage:** This is a basic functionality for MBaaS, but it can have advanced features like file storage and external DB usage capability. The offline capability is a big plus, the synchronization can be a hard task.
- **User management:** The user management is a basic feature too, it's connected with social media networks, it's a plus if the user can use their account of other sites. Beside the social media networks, LDAP can be used or OpenID. The login and registration is a common task, so it's a good idea to support it with customisable components on the client side.
- **Social media networks:** With the support of social media networks the users don't have to deal with the registration process. The most common social media sites are Google, Facebook and Twitter. Deeper integration is an extra, to use other features like Facebook graph search API.
- **Push notification:** The push notification support for iOS (Apple Push Notification Service) and Android (Google Cloud Messaging), the AB testing, client to client push notification capability and other extra features are valued.
- **Analytics:** The capability to track built-in user events like installing data, application start time and even custom events.

		Google Mobile Backend	Parse	Backendless	Amazon	Firebase	Kumulus	Kinvey
Pricing		free trial; pay/use	limited free version; pay/month	limited free version; pay/month	limited free version, pay/use	limited free version, pay/month	free developer version, pay/month	free developer version, pay/month
Host		by provider	by provider	by provider, hybrid, or custom	by provider	by provider	by provider	by provider or custom
Open source, license		no	no	no	no	no	no	no
Custom business logic		yes	yes	yes	yes	-	n/a	yes
Scheduled task		yes	yes	n/a	n/a	-	-	n/a
Server side language	**JavaScript**	-	yes	-	-	-	n/a	yes
	Java	yes	-	yes	yes	-	n/a	-
	other	python, php, go	-	-	-	-	n/a	-
Admin site		2	2	2	1	1	1	1
SDKs	**Android**	yes	yes	yes	yes	yes	yes	yes
	iOS	yes	yes	yes	yes	yes	yes	yes
	JS	yes	yes	yes	-	yes	yes	yes
	other	-	.NET, PHP, python, unity, C, etc	.NET, ActionScript	fireOS	-	PHP, Unity	-
SDK		1	2	1	1	1	1	1
IoT		2	2	0	1	1	0	1
Documentation		2	2	2	1	1	1	2
Tutorials		2	2	1	1	1	1	2
Sample apps		1	1	1	1	2	n/a	1
Storage		2	2	2	2	1	1	2
User management		1	2	1	1	1	0	1
Social media networks		yes, Google	yes, FB, Twitter, Google	yes, FB, Twitter	yes, Amazon, FB, Google	yes, Twitter, Google, FB	-	yes, FB, Google
Push notification		1	2	1	1	0	0	n/a
Analytics		1	2	1	2	0	1	0

		FeedHenry	Appcelerator	BaasBox	DreamFactory	StrongLoop	UserGrid	Helios
Pricing		limited free trial, pay/month	limited free version, pay/month	free	free	free	free	free
Host		custom	by provider, hybrid or custom	by provider or custom	by provider or custom	custom	custom	custom
Open source, license		no	no	yes, Apache 2	yes, Apache	yes, MIT license, or StrongLoop License	yes, Apache	yes, MIT
Custom business logic		yes	yes	yes	yes	yes	yes	n/a
Scheduled task		yes	yes	yes	n/a	n/a	n/a	
Server side language	**JavaScript**	yes	yes	yes	yes	yes	-	n/a
	Java	-	-	-	-	-	yes	n/a
	other	-	-	-	-	-	-	n/a
Admin site		2	2	1	1	1	1	1
SDKs	**Android**	yes	yes	yes	yes	yes	yes	-
	iOS	yes	yes	yes	yes	yes	yes	yes
	JS	yes	yes	yes	yes	yes	yes	-
	other	WP8, HTML5, Xamarin, Cordova, Appcelerator	Windows, Blackberry, HTML5	-	WP, Titanium	-	ruby, .NET, PHP	-
SDK		2	2	1	1	1	1	2
IoT		0	0	0	1	0	0	0
Documentation		2	2	1	1	1	1	0
Tutorials		2	1	1	1	1	0	1
Sample apps		1	1	1	1	n/a	0	0
Storage		2	2	2	1	1	1	1
User management		1	1	2	1	1	1	1
Social media networks		n/a	yes, FB, Twitter, LinkedIn	yes, FB, Twitter, Google	n/a	yes, FB, Google, Twitter	yes, FB	n/a
Push notification		1	1	1	1	1	n/a	1
Analytics		2	2	1	0	0	0	1

COMPARISON DISCUSSIONS

PaaS summary

All of the overviewed providers support the major development languages nowadays. Providers can be grouped in two categories based on their size. The bigger providers, like Google, Amazon, Microsoft, they have been in the PaaS market for some time, so they have their own solution, host on their own servers, integrate their solutions into their bigger infrastructures.

The smaller PaaS providers, (CloudControl, AppScale, Deis, Mendix, etc.) are trying to get some interesting solutions on the market, and have interesting ideas that are not available with the big providers, such as hosting on multiple cloud backends, run the PaaS service on the premise of the customer, or a new way of creating apps thereby shortening the time-to-market off the application. These smaller providers usually have their backends at larger providers, mostly at Amazon. This presents a dependency and a connection of reliability for them. If there is an outage with an IaaS provider that they use, than their product will be offline as well. Some of the providers (Mendix, AppScale) aim to provide an abstraction layer between different PaaS providers, with some kind of middleware or API system. The goal of this is to remove the dependency on a particular PaaS provider (to prevent vendor lock-in), and to make the applications more portable between providers.

Our recomandation is differs based on the goals, but in general we can say that with the Heroku provider we had really great overall experience. The Google App Engine should be considered as a good choice because of the other Google services which can be cooperated with each other, and this kind of Google ecosystem clearly has its own benefits. But every situation has different priorities, the opensourceness can be a big plus too in some cases for example.

MBaaS summary

Most of the researched MBaaS solutions have some common features, these features are the basic features, but they are still big categories. There are some extra capabilities for every feature that makes the solution special, leading in some area.

The storage is one of the basic features, it enables to store, modify, delete and load data to and from the cloud. It can have some non basic capability, like connectors to external DBs and systems, real time data sharing, offline working, data synchronization. The user management is a basic feature too, it can have extras like using other systems (LDAP, OpenID, OAtuh, FB, Google, Twitter) or the capability to "follow" other users. The social media network integration usually ends in the existing user account usage, but it can have full integration, for example the capability of using the Facebook SDK and the FB Graph API. The push notification can be used for chat messaging and A/B testing. The Analytics has key functionality to track the users and analyze them, the visualised data, and generated reports can help to accomplish these tasks. The developers are interested in crash reports, but the management probably more curious about the user behaviour. The other important feature is the server side customizability capability, which can have multiple levels, starting from database like data modification triggers to custom business logics and scheduled jobs. The most supported client side platforms are the Android, iOS and JavaScript. The support usually just a generated SDK, but it can have more, like UI components, utility modules. The viewpoint can be different, sometimes the cloud is the main thing, and the client side is just for minimal interaction and visualisation, but sometimes the client platform has the focus and the cloud side is just a helper tool. The admin site is important too, it has many features and options, but it must stay simple and user friendly.

The solutions can be categorised by the targeted developers for example. Some of them are for startups and indie developers, their goal is to create simple, fancy mobile applications as fast and efficient as possible and still with little money (at the beginning). Others are for big enterprise companies where the security, integration, analytics and monitoring are the key features. The other possible way to distinguish the overviewed solutions is based on their open source nature. At this point the open source projects are mostly in beta versions and only capable of basic features, but they have the advantages of the freedom of the deployment and the community support. Commercial providers usually provide SLAs.

Surprisingly many companies have MBaaS solutions. There is a big chance that the ones with very special skill sets can satisfy special needs and survive in the long run. But the universal, general solutions can have a bright feature too, but their MBaaS features are just one of the many other features to support the whole development, deployment and testing of the mobile and web development. These are can be called as Mobile Development Platforms. Most of the providers uses Amazon's cloud as hosting, and Amazon has its own MBaaS solution. But still there is space for many MBaaS solutions because of the specializations (targeted developers, real time capability, streaming, game features, stb.)

There are many PaaS providers, and much more different requirements for every project, so it isn't an easy job to deside which one is the best, but we thing that Parse is a good choice in most of the cases. It has a strong background (Facebook), evolves fast and it's really userfriendly. The Parse SDKs are all open source projects, and they are not afraid of innovation. Further looking the Google MBaaS solution is a great option considering the opportunities offered by Google. But every situation is different, if it's a small and simple project FireBase can be ideal in some cases, or BaaSBox has an adventage because of its open sourceness.

The biggest challenges are the security problems, the integrations problems and the offline working capability and the automatic data synchronization features.

SUMMARY

In this chapter we gave an overview of the state-of-the-art Platform-as-a-Service solutions that are used to develop applications over clouds. We investigated the basic capabilities of the most advanced providers: what levels of developer experience they offer, how they follow recent trends in cloud application development. Primary sources for this investigation were public documentation of the relevant tools, research publications and trial or demo versions where applicable. Developer experience was measured by creating and executing sample applications with some of these PaaS tools. We found that most of the examined providers are usable, and some of them have special strengths. We created comparison tables for highlighting the generally supported and lacking functionalities.

ACKNOWLEDGEMENT

The research leading to these results has received funding from Ericsson Hungary Ltd.

REFERENCES

Amazon Mobile. (2015. May). Forrás: http://aws.amazon.com/mobile/

Amazon Web Services. (2015. May). Forrás: http://aws.amazon.com/

Apcera Continuum. (2015. May). Forrás: https://www.apcera.com/continuum/

Appcelerator. (2015. May). Forrás: http://www.appcelerator.com

AppScale. (2015. May). Forrás: http://www.appscale.com/

Azure SDK for Node. (2015. May). Forrás: https://github.com/Azure/azure-sdk-for-node

Azure SDK for Python. (2015. May). Forrás: https://github.com/Azure/azure-sdk-for-python

Azure Web Job. (2015. May). Forrás: https://azure.microsoft.com/en-us/documentation/articles/web-sites-create-web-jobs/

BaasBox. (2015. May). Forrás: www.baasbox.com

Backendless. (2015. May). Forrás: www.backendless.com

Bluemix. (2015. November). Forrás: https://console.ng.bluemix.net/

Bosh-lite. (2015. May). Forrás: https://github.com/cloudfoundry/bosh-lite

ClearDB. (2015. May). Forrás: https://www.cleardb.com/

CloudControl. (2015. May). Forrás: https://www.cloudcontrol.com/

CloudFoundry. (2015. May). Forrás: http://cloudfoundry.org/

Cloudinary. (2015. May). Forrás: http://cloudinary.com/

Cloudinary SDK for Python. (2015. May). Forrás: https://github.com/cloudinary/pycloudinary

convert. (2015. May). Forrás: http://www.imagemagick.org/script/convert.php

David Cunha, P. N. (2013). A Platform-as-a-Service API Aggregator. In *Advances in Information Systems and Technologies* (old.: 807-818).

David Cunha, P. N. (2014). PaaS Manager: A Platform-as-a-Service Aggregation. *Computer Science and Information Systems, Vol. 11, No. 4*, 1209–1228.

Deis. (2015. May). Forrás: http://deis.io/

DreamFactory. (2015. May). Forrás: http://www.dreamfactory.com/

Dzone. (2015. May). *Guide to Cloud Development*. Forrás: http://dzone.com/research/2015-guide-to-cloud-development

Express. (2015. May). Forrás: http://expressjs.com/

FeedHenry. (2015. May). Forrás: http://www.feedhenry.com/

Firebase. (2015. May). Forrás: www.firebase.com

Foreman and Procfile. (2015. May). Forrás: http://ddollar.github.io/foreman/

gevent. (2015. May). Forrás: http://www.gevent.org/

Google App Engine. (2015. May). Forrás: https://cloud.google.com/appengine/

Google Endpoints. (2015. May). Forrás: https://cloud.google.com/mobile/endpoints/

Google Mobile Cloud Platform. (2015. May). Forrás: https://cloud.google.com/solutions/mobile/

Helios. (2015. May). Forrás: http://helios.io/

Heroku. (2015. May). Forrás: https://www.heroku.com/

Heroku hosted Redis. (2015. May). Forrás: https://addons.heroku.com/heroku-redis

Hoefer, C., & Karagiannis, G. (2010). Taxonomy of cloud computing services. *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, (old.: 1345 - 1350). Miami, FL.

ImageMagick. (2015. May). Forrás: http://www.imagemagick.org/script/index.php

Kolb S., W. G. (2014). Towards Application Portability in Platform as a Service. *Service Oriented System Engineering (SOSE), 2014 IEEE 8th International Symposium on*, (old.: 218 - 229). Oxford.

Loopback. (2015. May). Forrás: http://loopback.io/

Luis Rodero-Merinoa, L. M. (2012). Building safe PaaS clouds: A survey on security in multitenant software platforms. In *Computers & Security* (old.: 96–108).

MBaaS wikipedia. (2015. May). Forrás: http://en.wikipedia.org/wiki/Mobile_Backend_as_a_service

Memcache. (2015. May). Forrás: http://memcached.org/

Mendix. (2015. May). Forrás: https://www.mendix.com/

Michael Armbrust, A. F. (2009). *Above the Clouds: A Berkeley View of Cloud.*

Microsoft Azure. (2015. May). Forrás: http://azure.microsoft.com/

MongoDB. (2015. May). Forrás: https://www.mongodb.org/

MySQL table deletion workaround. (2015. May). Forrás: http://stackoverflow.com/questions/2300396/force-drop-mysql-bypassing-foreign-key-constraint

Node.js. (2015. May). Forrás: https://nodejs.org/

Node.js HTTP module. (2015. May). Forrás: https://nodejs.org/api/http.html

node-redis. (2015. May). Forrás: https://github.com/mranney/node_redis

OpenShift. (2015. May). Forrás: https://www.openshift.com/

Parse. (2015. May). Forrás: www.parse.com

phpMyAdmin. (2015. May). Forrás: http://www.phpmyadmin.net/home_page/index.php

Pillow. (2015. May). Forrás: https://python-pillow.github.io/

Pivotal. (2014. August). *PaaS comparison*. Forrás: http://www.slideshare.net/Pivotal/paa-s-comparison2014v08

PostgreSQL. (2015. May). Forrás: http://www.postgresql.org/

Protocol Buffer. (2015. May). Forrás: https://github.com/google/protobuf/

PyMongo. (2015. May). Forrás: http://api.mongodb.org/python/current/

Python. (2015. May). Forrás: https://www.python.org/

Rajkumar Buyyaa, C. S. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. In *Future Generation Computer Systems* (old.: 599–616).

Redis4You. (2015. May). Forrás: http://redis4you.com/

Rimal, B., Choi, E., & Lumb, I. (2009). A Taxonomy and Survey of Cloud Computing Systems. *INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on*, (old.: 44 - 51). Seoul.

Sellami M., Y. S. (2013). PaaS-Independent Provisioning and Management of Applications in the Cloud. *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*, (old.: 693 - 700). Santa Clara, CA.

Sellami, M., Yangui, S., Mohamed, M., & Tata, S. (2013). PaaS-Independent Provisioning and Management of Applications in the Cloud. *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*, (old.: 693 - 700). Santa Clara, CA.

Sequelize. (2015. May). Forrás: http://docs.sequelizejs.com/en/latest/

Socket.io. (2015. May). Forrás: http://socket.io/

SQLAlchemy. (2015. May). Forrás: http://www.sqlalchemy.org/

Supervisord. (2015. May). Forrás: http://supervisord.org/

Twelve-Factor app. (2015. May). Forrás: http://12factor.net/

UserGrid. (2015. May). Forrás: http://usergrid.incubator.apache.org/

Wand. (2015. May). Forrás: http://docs.wand-py.org/en/0.4.0/

Yanpei Chen, V. P. (2010). *What's New About Cloud Computing Security?*