Migrating Existing PHP Web Applications to the Cloud

Ionuţ VODĂ
Zitec COM srl, Bucharest, Romania
ionut.voda@gmail.com

The purpose of this paper is to present a set of best practices for moving PHP web applications from a traditional hosting to a Cloud based one. PHP applications are widespread nowadays and they come in many shapes and sizes and that is why they require a special attention. The paper goes beyond just moving the code in the Cloud and setting up the run-time environment as some architectural changes must be done at application level most of the time. The decision of how and when to make these changes can make the difference between a successful migration and a failed one. It will be presented how to decouple and scale an application, how to scale a database while following the high availability principles.
Keywords: *PHP, Cloud Computing, Cloud Migration, Web Applications*

1 Introduction

Over the last decade web applications started to become more and more popular. Among various technologies that powered these websites PHP is, with no doubt, one of the most popular. Most of the high traffic sites and platforms such as Yahoo.com, FaceBook.com, Wikipedia.com, Wordpress.com are using PHP to power their services [1]. Running on more than 200M active sites [2] it quickly became used on various Cloud platforms as well, as the need for scale and availability emerged for those sites.

While each Cloud vendor has documentations aimed to help developers and/or system architects to move their applications to Cloud, they are focused mostly on their services and they not provide a Cloud wide guidance. This can make the process of selecting a particular Cloud vendor a little difficult. The problem becomes even trickier for custom solutions that not always fit on a Cloud offering in terms of services they provide.

The contribution of this paper is in the way it presents migration strategies tailored for PHP applications, whether they are custom software or an open source platform.

The ultimate goal of this paper is to assist PHP developers, the system architects and eventually a company, when moving from a traditional hosting to Cloud, by helping them to choose the Cloud platform and services best suited for their applications.

2 When to migrate to Cloud

Every application can be migrated to Cloud, there is no doubt about it. However certain applications make a best fit for the Cloud in what regards the Cloud services utilization (such as: encoders [3], CDN [4] [5], relational databases [6] [7]) and automatic scaling of resources [8]. Others applications can be just moved to the Cloud without getting to fully leverage the resources a Cloud provide. The later applications are said to be less ideal candidates for a Cloud Migration. Each of the two categories will be described in more detail below.

Application types that make a good fit for a Cloud migration:

Applications with periodic and demanding processing needs: these can be applications that do batch analyzes on various data and which may become very computing intensive. Batch analyzes might refer to: file conversion from one format to another, reporting, semantic text analyzes, text indexing, data clustering, data classification, neural network training for machine learning and so on. All of these operations are time predictable, in what concerns their occurrence, and with a high demand for computing power which translates in a greater need for hardware. Although the hardware might be bought it might not worth the investment as the processing takes a limited number of hours per time unit (day, month and so on). More than that, a company can decide at a certain point to further accelerate the

DOI: 10.12948/issn14531305/18.4.2014.06

batch processing process so it can end more quickly, let's say in less than an hour as opposed of 3 or 4 hours. This would be very easy in a Cloud environment just by spinning up some more computing units, while it can become challenging if not impossible otherwise in a standard hosting due to hardware limitation.

Applications with unexpected usage peaks: these are applications exposed to outside world mostly (online shops, product sites, blogs etc) which have a steady user base that can do something at some point which can end up with attracting a very large number of users. For instance it can get referenced on a popular web page or they can host a video/audio track that gets viral [9] or release a marketing campaign that can become more successful than anticipated (coupons, promo codes, ad words, etc.). In other words these are applications that cannot predict the inbound traffic volume and the exact time frame but they have to handle it properly. They know the traffic will come in but they don't know how much, when and how it will be distributed. Just like the above category, hardware acquisition might not worth the costs due to fragmented usage which can lead to a lot of hardware just being sit there unused.

Applications faced with the need for High Availability (HA) and scalability: here are two categories: on one hand, there are applications that want to provide a HA service to their end-users and applications that expect an increase in their usage or plan to extend their services to a larger user base. A HA [10] setup means that the application has no single point of failure (SPOF) which is achieved by adding a redundancy to each endpoint, component or service that it has. Naturally this assumes for more hardware plus the mechanisms needed to achieve a HA such as: automatic fail-over, load balancing and monitoring. The second category covers applications that want to increase their usage base which, for instance, are extending their product or service offering or are expanding to new markets, to new countries and so on. This also comes with the need for new hardware and, again, the management layer on top of it to make it work.

Applications with the need for various testing environments: beside moving or running the production environment in the Cloud, which we covered above, one can choose to use the Cloud for running a test environment. The test environments can be permanent, in case we are talking about a staging [11] layer or temporary when just a specific set of tests wants to be performed, such as security testing [12] or load testing [13]. Either one, some hardware is again needed. The cost impact in hardware acquisition is much higher for the latter category (security testing and load testing) as these tests can be performed a few times a year. The problem with them is that they require a full blown production setup in order to be relevant and that is why getting all that hardware might become inefficient from a cost perspective.

Like mentioned above there are some applications that, while they can be moved to Cloud as well, they will not fully benefit the Cloud components. The applications falling under this category are mostly intranets like: invoicing software, CRMs [14], ERPs [15] and so on. Their main characteristics is that they have a predictable user base (the employees) and usage pattern (working hours). This predictability and constant usage can give room for a company to plan ahead and buy for its own hardware. Of course that at some point these applications too can become good candidates for the Cloud, but most of the times they do not reach the Cloud so quickly.

Ultimately the Cloud offers a pay-as-you [16] go model and managed services (auto-scaling, database services and so). These two are very attractive for companies that do not want to or afford to invest in a full blown infrastructure for their services. Most of the companies prefer to keep the focus on their core business and let a Cloud vendor deal with the rest.

3 Cloud types

Depending by the level of management for their services, the Clouds can be classified in three types: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). With IaaS standing at the bottom, PaaS on top of it and SaaS above the

DOI: 10.12948/issn14531305/18.4.2014.06

two, this is also referred as "Cloud pyramid" [17], Figure 1.

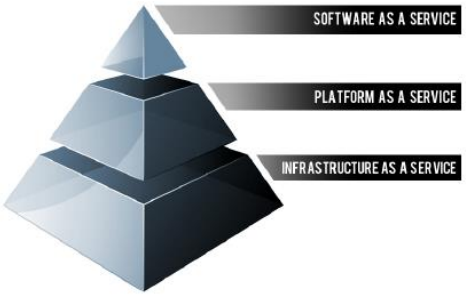

Fig. 1. The Cloud pyramid [17]

IaaS, as the name suggest, delivers cloud infrastructure such as: computing (through virtual machines), queues, storage, load balancers, VLANs, operating systems and others. It requires in depth technical knowledge in order to deploy and operate an application. Each component has to be managed: scaled up/down and load balanced. As IaaS computing units are in fact virtual machines (VMs), this gives a total freedom of what software can be deployed and how it can be configured. However it is in user responsibility to patch and update the running software.

Less appealing with IaaS is the monthly cost calculation which can become very complicated when more and more services are being used in an application. Almost each service within a Iaas Cloud have at least two dimensions or attributes that account for billing. For instance computing units (i.e EC2 in AWS [18] or VM in Azure [19]) are being charged at operating hours and used bandwidth, a data storage service (i.e S3 in AWS [20] or Storage for Azure [21]) for storage and bandwidth and so on.

PaaS makes a step further and provides managed services. That means the software updates and patching responsibility falls to Cloud vendor. Another benefit when running PaaS is that the scaling, backup, data replication and other infrastructure specific operations happen transparently for the end user being orchestrated by the Cloud itself. Examples of managed services under PaaS are: SQL Database from Azure [7] which is a managed Microsoft SQL Server that comes with built in scalability, backup, ego-replication and software update management or Amazon DynamoDB [22] which is a fully managed NoSQL [23] database running in the Cloud. Just like SQL Database in Azure, the whole infrastructure management operations are taking place transparently for the end-users.

SaaS sits on top of the pyramid and it comes with fully working software being managed in the Cloud. A SaaS application can encompass both IaaS components and PaaS applications. It can use either of the two separately of course. Examples of SaaS software are video platforms such as (YouTube, Vimeo), email services (Gmail, Yahoo Mail) or CRM software (Microsoft Dynamics CRM, Zoho).

As noted, IaaS requires most of technical knowledge between the three. With it, one will probably need a developer to run an application (or at least make it run) or to migrate it in the Cloud. By moving to PaaS a little more comfort is gained as much of the maintenance work is carried by the Cloud itself. However at this level some technical know-how is still needed to make an application work. SaaS is the easiest to adopt as it requires no infrastructure configurations. It however requires software configurations like any other application.

Nowadays the boundaries of Cloud platforms are becoming very thin as most of the vendors are exposing mixed services that fall either under IaaS, either under PaaS. Amazon Web Services started with IaaS offerings (EC2, SQS and S3) and added PaaS components along the way (relational databases and non-relational databases). Microsoft Azure offers both PaaS components (Azure Websites [24] and SQL Database) and IaaS (Virtual Machines and Storage).

4 Planning a cloud migration

A cloud migration consists in several distinct steps each with its unique characteristics. Before going through these steps, it is assumed that the application or at least the modules/components of the application that want to be migrated to Cloud were identified. The steps to be undertaken for a migration are as follows: determine the architecture blueprint,

DOI: 10.12948/issn14531305/18.4.2014.06

choose a Cloud vendor, refine the application architecture, testing the setup and the release.

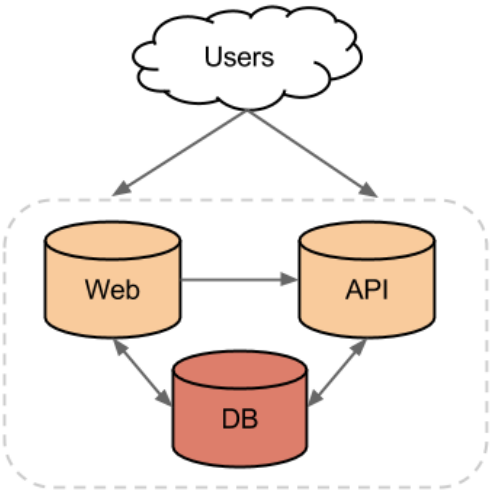

Fig. 2. The initial application running on a single server

We will use as an example a custom PHP application sitting on one server running with three components: a front-end layer, an API [25] and a data persistence layer, which is a MySQL database [26] in our case. Before moving it to Cloud the application components look this way:

Determine the architecture blueprint: in this very first step the scalable components of the application or part of the application that need to be migrated to Cloud are identified. This step should be Cloud independent in the way that the components should not rely at this stage on a particular Cloud service or offering. Of course if a company have already chosen the Cloud vendor, it can include specific components at this stage too. The main point of this step is to give one an idea of the scale of migration and to consist as a base for cost calculation (the immediate next step). At this stage, the initial setup would transform like indicated in Figure 3.

Choose a Cloud vendor: once the independent and scalable parts of the application and the software running on each of them were identified, the next step would be to find a best fit for them among the Cloud vendors. In a perfect world, each of the part of the application would go on a Saas or PaaS component.

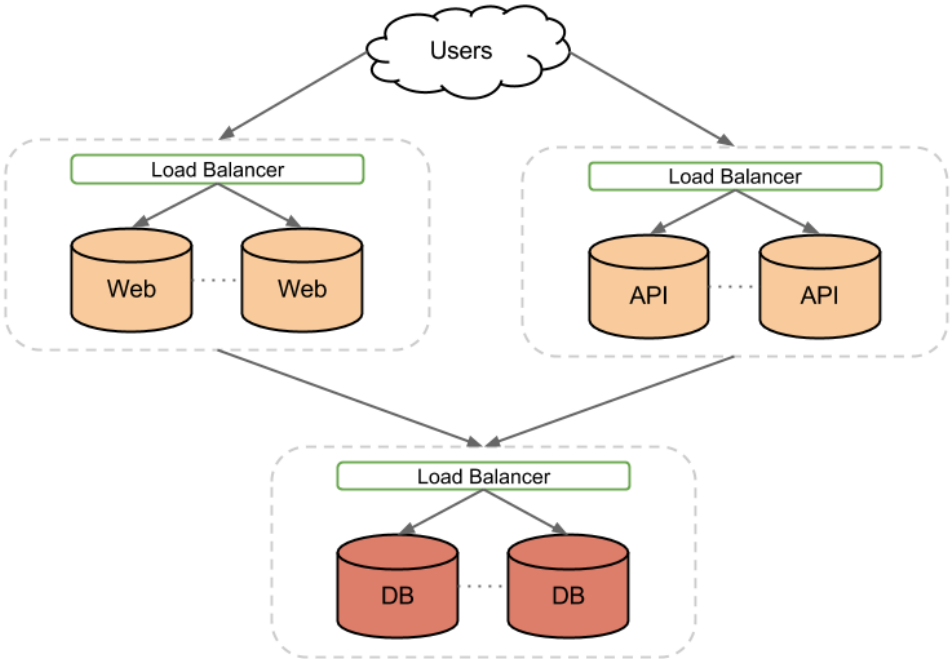

Fig. 3. The blueprint architecture, Cloud independent

For instance a SQL Server database will be a best fit for Azure SQL Database [17] or on AWS RDS [6], while a MySQL database could go to AWS RDS too. In case the application would long term durable storage for infrequently accessed data, then Amazon Glacier [27] would be a good option.

DOI: 10.12948/issn14531305/18.4.2014.06

Also, at this stage, a preliminary cost for each Cloud vendor should be calculated. Each vendor has online price calculators for that such as Azure Price calculator [28] or AWS Simple Monthly Calculator [29]. When doing a price calculation, multiple things must be taken into account, starting from obvious ones such as computing hours, bandwidth and storage and ending up with less obvious metrics, like: number of GET requests for accessing a certain object storage, load balancer hours, provisioned IOPS [30] for certain storage options and many more.

Our example setup requires: computing hours, persistent storage, bandwidth and load balancer hours. Also, since we are using a MySQL database, AWS will be a good fit as a Cloud solution, by using their Relational Database Services (RDS) [6]. Of course that AWS is not the only option but the best given our example and for the purpose of this exercise.

Refine the application infrastructure: like already mentioned, the Cloud vendor impacts the final architecture of the application. Figure 4 shows how the blueprint architecture changed to accommodate Cloud specific components. Like shown, new services were used such as DynamoDB [22] for PHP session storage, S3 [20] for logs storage and RDS for MySQL database, which allows a master-slave architecture with a hot standby master.

Once the final architecture was set in place, a new and final cost calculation should be done.

Testing the setup: the most important thing to do when testing a setup is to consider real size data. That means one should aim to do a complete clone of the production data, which mostly includes database data and user generated content.

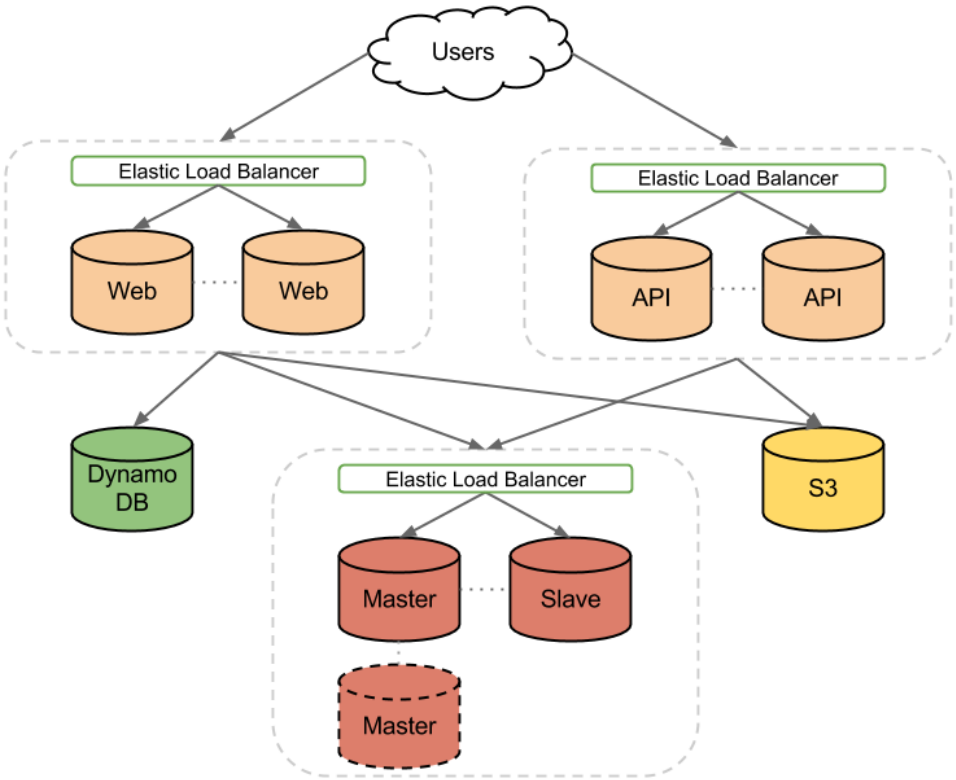

Fig. 4. The final architecture, Cloud specific

A database can be imported in several ways depending by the database engine: a full dump from the source database with an import into source database, by using the binary logging or by setting the source database as an external master and the destination database as slave [31]. The later method is the most convenient

DOI: 10.12948/issn14531305/18.4.2014.06

way to do as it will eventually have no downtime when switching from one database to another. However not all database engines and Cloud vendors offer this options. When using both binary logging and external master import options, the sequence of operations is as follows: first an initial dump is created from the source database. It contains both database schema and data. This initial dump is also called a baseline. Second, the binary logging is activated on the source database server, which will indicate it to start doing incremental log files, which contains changes past the moment of baseline creation. Third, the baseline is imported on the destination database server. Forth, the incremental log files are imported so the destination database can be tested. Once the testing is ready, the destination database can be dumped, the baseline reimported and the log files applied. Care must be taken with imported data during testing in what concerns user sensitive data such as: phone numbers, email addresses and so one. Extra care must be taken when the application issues emails, texts messages or even phone calls automatically based on certain events. It is recommended to alter these sensitive data by concatenating them with a random string.

The user generated content can be imported either in batch mode or continuously. Batch mode refers to moving blocks of content from source server to destination server. The downside with this approach is that it doesn't account for data changes on the source server. So, every time something changes on the source server, everything needs to be copied on the destination server as well. The continuous method assumes an initial data sync between the two servers. To achieve that an application needs to be installed on both ends, such as Rsync [32]. Then the application will take care of synchronizing data changes between the two servers when changes occur on the source server.

Release: this is the moment when the tested setup becomes promoted as a production environment. When talking about web applications, the release of such a setup mostly consist in a DNS change so that the application domain name to point to an IP address that belongs to the new setup. Attention must be paid with the DNS changes as they require some time to propagate. The TTL [33] of that specific record must be decreased to lowest value permitted by the name server of that specific domain. Lower the DNS TTL, faster the changes will propagate through the internet.

5 Migration scenarios for PHP applications

Although the applications should be decomposed into decoupled components and ran on scalable infrastructure, like shown in Figure 4, there are cases when a certain company does not want this from the very beginning. For instance when they want to simply run a simplified testing setup in the Cloud or when the company tests that specific Cloud. Another special case is with PHP platforms such as Wordpress, Drupal or Joomla that can be hosted in PaaS offerings of different Clouds.

Depending on how complex and decoupled the application setup will be, a few infrastructure scenarios emerge:

Platforms: certain PHP platforms like Wordpress, Drupal or Joomla can be hosted in PaaS offerings of certain Cloud vendors. This makes it very appealing for site owners to move their sites to Cloud. Depending of the level of customization that a platform site has, it can be moved faster or slower to Cloud. Greater the number of customizations, higher the time to migrate it to Cloud and little the chances to work “out of the box” on a PaaS offering. For instance, Azure Websites offers PaaS for these platforms: Wordpress [34], Drupal [35] and Joomla [36]. PaaS is the best choice for site owners as it offloads completely the infrastructure maintenance. Backups, replications and availability are all handled by the Cloud vendor. This setup can be used for production environment.

Single virtual machine scenario: this is the most basic setup of all and consists in replicating the off-Cloud deployment on a virtual machine. This assumes that the whole application, before Cloud migration, runs on a single server. This kind of setup is mostly used when setting up a development environment or when running a isolated test on the cloud like

DOI: 10.12948/issn14531305/18.4.2014.06

security testing or performance testing. There is no HA [10] associated with this setup and should not be used for production environment.

Each layer on a virtual machine: this goes a little further from the single virtual machine approach, by moving each component on its own layer. For an application consisting in front-end and database, this means to put each one on a virtual machine. In case the application has other compute intensive components such as search engine, documents encoding, etc. they can be as well moved to dedicate virtual machines. Although a separation of the layers is achieved, there is no HA with this approach. Each service is a single point of failure. This setup should not be used for production environments but for testing only. It is a good step towards a fully scalable and HA setup but it should not be treated as one so far.

Full blown HA setup: with this approach each component is isolated on a scalable and highly available group, just like already explained in chapter 4. Planning a cloud migration. Figure 4 indicates such a setup.

6 Database considerations

Scaling the database can be done in different ways depending of the usage type that it has.

A read intensive database can be scaled in a master-slave schema (see Figure 5). It assumes for one master server taking the write queries (insert, delete, update, alter) and one or multiple slaves serving read queries (select). The replication between master and slaves can go either synchronously or asynchronously. With former the master responds to the client when the changes propagated on all nodes, while with the later it can respond immediately and the changes propagates to slaves in background. In order to avoid a single point of failure for the master server, a hot standby server is being used. It receives all data and schema changes, just like a slave would do, but without being actually used by the application. The hot standby will be used only when the primary master becomes unresponsive. It is the application concern to handle this switch. In case of Amazon RDS this switch is automatically handled by the system.

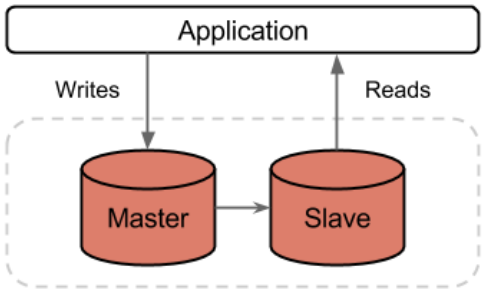

Fig. 5. Database master-slave setup

A write intensive database needs more masters, or more servers taking writes, than slaves. Two common approaches [37] are used in this scenarios: a share nothing model [38] (also known as data partitioning [39]) where each node has its own sub-set of data and the share everything model (also known as shared-disk or master-master replication [40]), where each node has the same data set and thus is able to handle both writes and reads. Each database engine has its own way of dealing with data partitioning (see Figure 6), while for a master-master replication (see Figure 7) there are usually third party applications handling that.

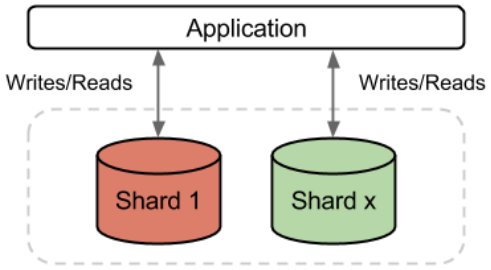

Fig. 6. Database partitioning

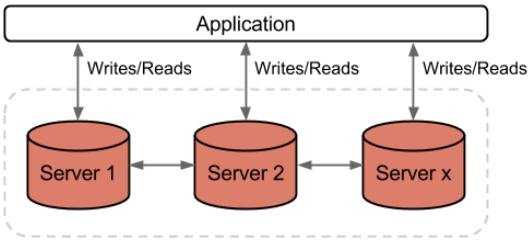

Fig. 7. Database master-master setup

ScaleDB [41] or Percona XtraDB [42] are commercial solutions for handling master-master replication for MySQL. Scaling on writes is known to be more problematic than scaling on reads as input data needs to be synchronized among the nodes. CAP theorem

DOI: 10.12948/issn14531305/18.4.2014.06

[43] constrain is a factor that should also be considered when scaling writes. Of course that the scalability issues we covered above, in what concerns the databases, applies when using IaaS Clouds to deploy and run the database. In case of a PaaS, that would be handled by the Cloud vendor seamlessly. The scaling strategies depicted in Figure 6 and Figure 7 can also be used for scaling a read intensive database or a mixed (reads and writes) database. However a master-slave approach comes more in handy for scaling read intensive databases.

7 High Availability considerations

The most important thing to achieve high availability is to separate the components and add redundancy to each one of them. This concept was strongly enforced along the paper. The next most important thing, once the application was decoupled, is to decide where the redundancies will be placed from a physical location perspective. They can stay in the same data-center, in the same geographical region or, in the ideal case, they can be replicated across geographical zones.

When the application is contained within a single data-center, there are chances that the entire facility to end up with a major malfunction, caused by an external event such as an electrical storm, power supply loss or other. This poses a great risk for an application even though it has been deployed by following the HA principles. Another risk is that the data-center could reach its maximum resource capacity (i.e it can no longer start new computing power) which can limit the application auto-scaling capacity.

The next level is to have the applications hosted in two different data-centers within the same geographical area. Usually the data-centers are in a range of tens of kilometers away of each other. While obviously better than the single data-center option, it poses almost the same risks as the other one. It already happened for both Amazon [44] [45] and Azure [46] [47] to have one or more zones affected in a single incident.

To reach the ultimate level of high availability each the application must run on at least two geographical areas. This model comes with greater costs of course, as the entire application setup must be replicated to another region, and with a greater maintenance overhead. Not all IaaS Clouds offer geographical region data replication for all of their services. Each vendor should be consulted for this specific topic by visiting their infrastructure pages: Azure Regions [48], AWS Regions and Availability Zones [49] and Rackspace global infrastructure [50]. So, choosing where to host the application from a location perspective is combination of availability, implementation time and costs and maintenance overhead and costs. By implementation we mean the time to make the application setup running in the Cloud.

8 Things to consider

The following points should be carefully considered before moving an application to Cloud, ideally in planning phase of the migration. A part of them are PHP specific, others to distributed computing model and a part of the Cloud vendor specific.

Eventual consistency model: the eventual consistency model [51] is specific to distributed applications running on multiple nodes. It affects almost all distributed applications and services. Although the nodes are transparent for the Cloud consumer, they can impact the data propagation and the Cloud response codes for certain services. To better illustrate the concept, we will assume that an application uploads files to a distributed storage such as Amazon S3 or Azure Storage. Although the given storage system will accept the uploaded file, it might not be immediately available for reads or it might be shown inconsistent to subsequent read requests. That is because on the underlying nodes of that service, the file still gets propagated to all nodes. The eventual consistency model ultimately impacts the way the application works as it has to handle retries for both accessing an object or for uploading/sending an object to a given Cloud service.

PHP sessions storage: by default the PHP sessions are stored on the local file-system. This

DOI: 10.12948/issn14531305/18.4.2014.06

approach works fine for a single server running application, but it becomes unusable for two or more machines. In this scenario a user session will not propagate automatically to each server, which will make the application unable to recognize user stored data among subsequent requests. That is why a third party storage should be used for storing the PHP sessions. This could be either a simple key-value store or they can be stored in the database itself. Whatever the option, it should be highly available and reachable by all the application components that interact with session data. Almost each Cloud vendor offers key value stores, for instance: Azure Tables [53] or AWS DynamoDB. The size of the session file is a factor that should not be ignored as it can influence storage capacity and/or bandwidth throughput which will eventually reflect in costs. It should be even more considered in application with frequent session access.

Code deployment or release: another important aspect which changes when running in Cloud is the code deployment method. It becomes even trickier with a large number of machines in the range of tenths or hundredths. A manual code update is still manageable when it should be rolled on a single machine or even on two or three. As PHP does not come as a packaged application or at least not by default, multiple files could be updated simultaneously on different machines where the application is running on. To achieve that, one should employ a centralized system (that should obey the same availability and scalability rules as discussed so far) for machine and code management. This system serves for two purposes: to be able to push code updates on all running machines (when doing a code release) and to allow the new machines, started as result of scaling process, to update themselves with the latest code version. From an architectural perspective, this system has two components: a central point that keeps track of running machines and of the code versions each one is running on and a service or daemon deployed on each supervised machine. The later just receives commands from the central point and executes them. Some of the Cloud vendors offer such an application management service, like AWS OpsWorks [54] for instance.

9 Conclusions

The paper started by presenting a selection of the most important scenarios that could qualify an application for a Cloud migration. Then it presented the main cloud stacks (IaaS, PaaS and SaaS) and highlighted a few representative services from each one of them. It then went through planning stage which is an essential part of any cloud migration process. Here is the part where the application components are identified, decoupled and placed into highly available, independent and auto scalable groups. Next specific guidelines for migrating PHP applications were provided by covering some of the most popular platforms like WordPress, Drupal and Joomla. Moving forward then with database considerations and presenting three of the most popular database scalability approaches. MySQL database was chosen as case study because it was part of the so called LAMP [55] stack that became very popular in early 2000s. A significant part of these MySQL powered small sites evolved into larger sites and hence our attention to it. Some high availability considerations were also provided and ended up with a few things to be considered before moving an application to Cloud.

The contribution of this paper is in the way it covers both cloud generic concept and PHP web application specific use cases. It is Cloud neutral and focused on medium to large PHP sites with emerging computing and scalability needs. Less focus is put on enterprise level systems as they are less dynamic in what regards the Cloud adoption. The practical examples of moving database and user generated content from a standard hosting to Cloud should come in handy for sites and applications looking to migrate.

References

[1] Programming languages used in most popular websites, http://en.wikipedia.org/wiki/Programming_languages_used_in_most_popular_websites

DOI: 10.12948/issn14531305/18.4.2014.06

[2] PHP just grows & grows, http://news.netcraft.com/archives/2013/01/31/php-just-grows-grows.html
[3] Amazon Elastic Transcoder, http://aws.amazon.com/documentation/elastic-transcoder/
[4] Amazon CloudFront, http://aws.amazon.com/cloudfront/
[5] Azure CDN, http://azure.microsoft.com/en-us/services/cdn/
[6] Amazon Relation Database Services, http://aws.amazon.com/rds/
[7] Azure SQL Database, http://azure.microsoft.com/en-us/services/sql-database/
[8] Amazon Auto Scaling, http://aws.amazon.com/autoscaling/
[9] Viral Video, http://en.wikipedia.org/wiki/Viral_video
[10] High Availability, http://en.wikipedia.org/wiki/High_availability
[11] Staging site in software developmnet, http://en.wikipedia.org/wiki/Staging_site
[12] Security testing, https://www.owasp.org/index.php/Web_Appliction_Security_Testing_Cheat
[13] Load testing, http://en.wikipedia.org/wiki/Load_testing
[14] Customer Relationship Management (CRM), http://en.wikipedia.org/wiki/Customer_relationship_management
[15] Enterprise Resource Planning (ERP), http://en.wikipedia.org/wiki/Enterprise_resource_planning
[16] Pay as you go model, http://azure.microsoft.com/en-us/offers/ms-azr-0003p/
[17] Understanding the Cloud Computing Stack: SaaS, PaaS, IaaS, http://www.rackspace.com/knowledge_center/whitepaper/understanding-the-cloud-computing-stack-saas-paas-iaas
[18] Amazon EC2, http://aws.amazon.com/ec2/
[19] Azure Virtual Machines, https://azure.microsoft.com/en-us/services/virtual-machines/
[20] Amazon S3, http://aws.amazon.com/s3/
[21] Azure Storage, https://azure.microsoft.com/en-us/services/storage/
[22] Amazon DynamoDB, http://aws.amazon.com/dynamodb/
[23] NoSQL databases, http://nosql-database.org/
[24] Azure Websites, azure.microsoft.com
[25] Application Programming Interface, http://en.wikipedia.org/wiki/Application_programming_interface
[26] MySQL database, www.mysql.com
[27] Amazon Glacier, aws.amazon.com
[28] Azure price calculator, http://azure.microsoft.com/en-us/pricing/calculator/
[29] AWS price calculator, http:// calculator.s3.amazonaws.com/index.html
[30] Input/Output Operations Per Second, http://en.wikipedia.org/wiki/IOPS
[31] Importing Data From a MySQL Instance Running External to Amazon RDS, http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/MySQL.Procedural.Importing.NonRDSRepl.html
[32] Rsync, http://rsync.samba.org/
[33] TTL for DNS records, http://en.wikipedia.org/wiki/Time_to_live#DNS_records
[34] How to run Enterprise Grade WordPress sites on Azure Websites, http://azure.microsoft.com/blog/2014/05/13/how-to-run-wordpress-site-on-azure-websites/
[35] Migrating Drupal to Azure Websites, http://azure.microsoft.com/en-us/documentation/articles/web-sites-php-migrate-drupal/
[36] Created a Joomla website from the Gallery in Azure, http://www.arctg.com/blog/2251-create-a-joomla-website-from-the-gallery-in-azure.html?clcid=0x409
[37] Shared-Nothing and Shared-Disk database clustering architectures, http://www.scaledb.com/pdfs/ArchitecturePrimer.pdf
[38] Share nothing architecture, http://en.wikipedia.org/wiki/Shared_nothing_architecture
[39] Database partitioning, http://en.wikipedia.org/wiki/Partition
[40] Master-master database replication, http://en.wikipedia.org/wiki/Multi-master_replication
[41] ScaleDB, http://www.scaledb.com/

DOI: 10.12948/issn14531305/18.4.2014.06

[42] Percona XtraDB Cluster, http://www.percona.com/software/percona-xtradb-cluster
[43] CAP theorem, en.wikipedia.org/wiki/CAP_theorem
[44] Summary of the Amazon EC2, Amazon EBS, and Amazon RDS Service Event in the EU West Region, http://aws.amazon.com/message/2329B7/
[45] Summary of the Amazon EC2 and Amazon RDS Service Disruption in the US East Region, http://aws.amazon.com/message/65648/
[46] Summary of Windows Azure Service Disruption on Feb 29th, 2012, http://azure.microsoft.com/blog/2012/03/09/summary-of-windows-azure-service-disruption-on-feb-29th-2012/
[47] Windows Azure Service Disruption Update,http://azure.microsoft.com/blog/2012/02/29/windows-azure-service-disruption-update/
[48] Azure Regions, http://azure.microsoft.com/en-us/regions/#overview
[49] AWS Regions and Availability Zones, http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html
[50] Rackspace global infrastructure, http://www.rackspace.com/about/datacenters/
[51] Eventually Consistent – Revisited, http://www.allthingsdistributed.com/2008/12/eventually_consistent.html
[52] Virtualization, en.wikipedia.org/wiki/Virtualization
[53] Azure Table Storage, http://azure.microsoft.com/en-us/documentation/articles/storage-dotnet-how-to-use-tables/#what-is
[54] AWS OpsWorks, aws.amazon.com/opsworks/?nc2hl3_dm/
[55] LAMP stack, http://en.wikipedia.org/wiki/LAMP

Ionuţ VODĂ has graduated the Faculty of Transportations, Department of Electronics, from University Politehnica of Bucharest in 2007. He started to work in software development field since 2001 and focused on web development from 2003. His work involved various programming languages (PHP, C#, Java, Java Script and Python), database engines (MySQL, PostgreSQL, SQL Server, Simple DB, DynamoDB) and covered software from educational field, B2B e-commerce platforms, APIs, document ingestion and indexing, ending up with full stack video encoding and delivery platform. He is working with cloud platforms since 2008, when he started with AWS, continued with Rackspace Cloud and eventually with Azure. His latest research interests cover clustering, classification and natural language processing algorithms. Starting with 2006 he works at Zitec COM, a Romanian software outsourcing company, where he holds the position of Chief Technology Officer.

DOI: 10.12948/issn14531305/18.4.2014.06