Evolution of Handling Web Applications Up to the Current DevOps Tools

Christian A. Rodríguez, Lía Molinari, Fernando G. Tinetti
LINTI, III-LIDI, Facultad de Informática
Universidad Nacional de La Plata, CIC Provincia de Bs. As.
La Plata, Argentina
{lmolinari, car, fernando}@info.unlp.edu.ar

***Abstract*—In this paper, we show the way in which many standard web applications have evolved. Most of them started with a complete independence among software development and its architecture, deployment and infrastructure design. Now, the current usage of DevOps (for Development and Operations) emerging tools are used in the context of new methodologies. We use a specific application and its own evolution as an example. Most of the development cycle, application architecture, and environments for development, testing, and QA (Quality Assurance) has changed in the last years. From 2006 to date, we have taken advantage of many new technologies as soon as they were available, each one providing some aid to (some) web development task. We also define a preliminary characterization for web applications evolution in order to analyze pros and cons of the current DevOps methodologies.**

Keywords—DevOps, Continuous Delivery, Continuous Deploy, Infrastructure Management

I. Introduction

Transactional web applications consume resources as user concurrency increases. Proportional resource consumption per connection or user access is a recurrent *pattern* regardless of the language or framework in which the web application is implemented. Several approaches to avoid running out of resources have been used, such as: a) increase hardware and infrastructure resources (aka scale up), in either vertical or horizontal scalability [41], and b) assigning specific or independent resources to specific (overloaded) subsystems or components, such as databases.

We will use a CMS (Content Management System) developed in 2006 as an example for our study. The CMS is a now classical PHP-MySQL application, that began being the solution to a specific web site, and it was reused in/adapted to other several different web sites and web portals. This product was developed using Symfony [33] [35], and although we will take it as an example the same analysis could be applied to portals based on Wordpress [19] [42], Drupal [10] [13], or Refinery CMS [30], among others. The functionality of this type of application is based on two specific sub-systems, as shown in Fig. 1: one of public (generally massive) access, and another for content management, available for a few users in charge of editing articles, images, and define its visual style.

Currently, the business of many companies goes through the web processing such as CMS, customer services, or e-commerce [20]. This shift from business to e-business has imposed strong requirements on code development and availability of services, especially via in-production web sites and web applications, and web services. Enhancing service and functionality availability has been achieved by agile application development, semantic versioning of each candidate for production, application deployment, and web platform updates [2] [38]. While agile software development methodologies are now well established, the service downtime is minimized through practices provided by conventions, tools and, above all, communication among development and operations areas. Thus, the new term DevOps (as a combination of Development and Operations) has been coined [18]. DevOps tools are specifically required in environments involving replication where the same updating steps are required in several similar instances.

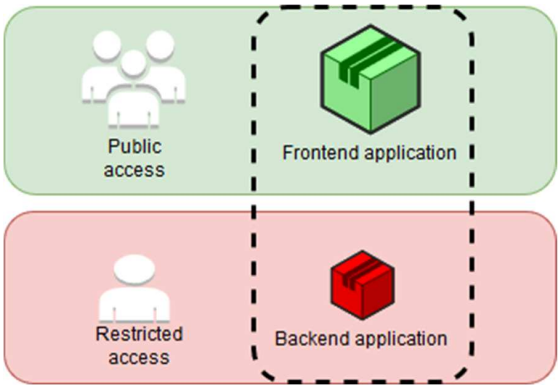

Fig. 1. Schematic View of a CMS Web Application.

As a monolithic application evolves to a composition of interacting services with high availability requirements, a project handling methodology becomes imperative. Then, the risk of introducing errors in production is minimized by strategies that involve development practices in terms of semantic versioning, testing, and continuous integration. In the operations area, concepts such as delivery and continuous deployment are fundamental to promote product versions in times consistent with agile development. In turn, specific infrastructure management strategies are needed, ranging from the deployment of web applications, configuration of servers using infrastructure tools such as those for configuration,

978-1-5386-2652-8/17 $31.00 © 2017 IEEE
DOI 10.1109/CSCI.2017.170

containers, and even the management of data centers as code.

II. Application Evolution

The main initial problem of the CMS was the number of concurrent users, which steadily grew through the years as shown in Fig. 2 (2017 contains only the first half). Fig. 3 also shows some well-defined peeks in two specific months every year. The supporting infrastructure had to be scaled up mainly due to two not necessarily independent growing factors: 1) software architecture, which grew including functionality (and some components were assigned specific exclusive resources), and 2) concurrent users, as shown in Fig. 2 and Fig. 3.

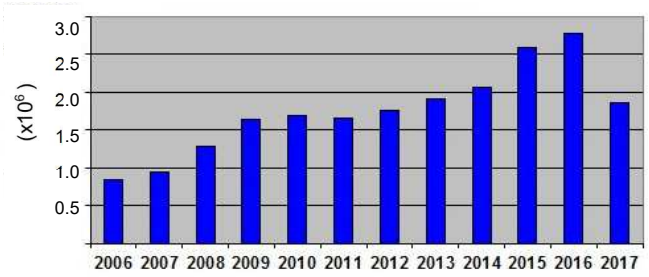

Fig. 2. Web Accesses per Year Since 2006.

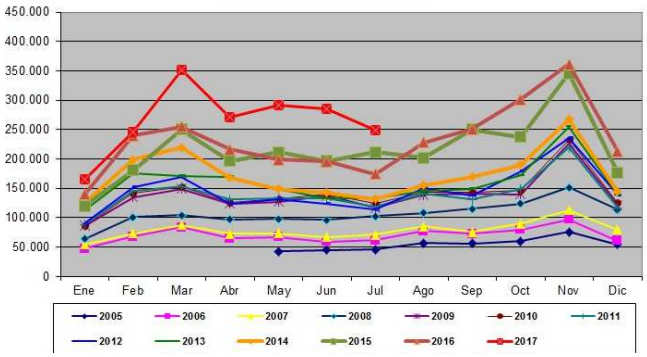

Fig. 3. Web Accesses per Month Since 2006.

The initial software architecture and its corresponding infrastructure allocation were monolithic, as shown in Fig. 4.

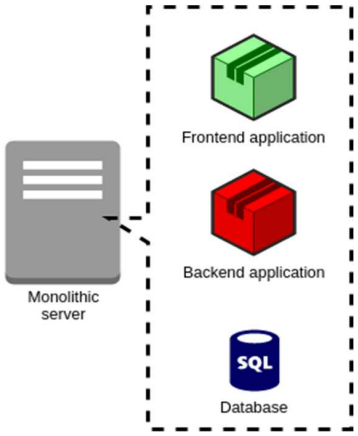

Fig. 4. CMS in a Monolithic Server.

The monolithic approach was not able to handle the growing number of concurrent accesses, and specifically the resources were not enough for the number of database connections. Three improvements were designed and implemented: 1) Use a database server, 2) Use FastCGI [24] for controlling and optimizing resources required for PHP processing, through PHP-FPM [26], and 3) Use lighter web servers to serve static content not requiring PHP processing. The first approach is a classical horizontal scale up, while the second and third are related to resource optimization. Furthermore, the new setting includes that concurrent accesses are rejected instead of letting the whole site unstable or unavailable.

A. Isolated Database and Horizontal Scale Up

The first improvement to the architecture and infrastructure shown in Fig. 4, is that of assigning a specific server for handling the database. Fig. 5 shows the independent server assigned to the database, thus unloading the server handling the public services, which in the previous configuration handled everything.

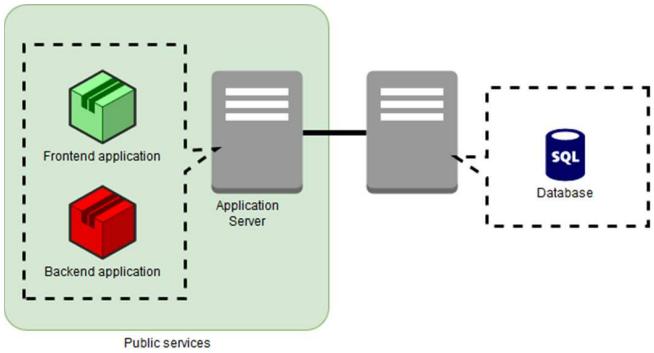

Fig. 5. Database Server with Exclusive Resources.

However, increasing the number of connections supported by the database was not enough for the increasing concurrent users' growth.

B. Horizontal Scale Up

At least at the conceptual level, handling more concurrent accesses by horizontal scale up is simple: clone or replicate application server/s. However, cloning application server/s leads to solving two *new* problems: a) sharing files that must be available in each instance, and b) use a load balancer for distributing workload among each replica. Fig. 6 schematically shows the *new* architecture and infrastructure with replicated application servers, file sharing, and a load balancer.

Maintaining web sessions was the next drawback related to horizontal scaling. Several session management alternatives were analyzed, such as: a) databases specifically devoted to session handling, b) Redis [29], and c) Memcached [22]. The latter was selected, and the system evolved to that shown in Fig. 7 (schematic view).

C. Further Improvements

Response times were significantly reduced by combining two tools: a) The load balancer is replaced by a caching HTTP reverse proxy, Varnish [16], and b) Use a CDN (content Delivery Network) for decoupling static data [37], so that

application servers are involved only with dynamic content processing.

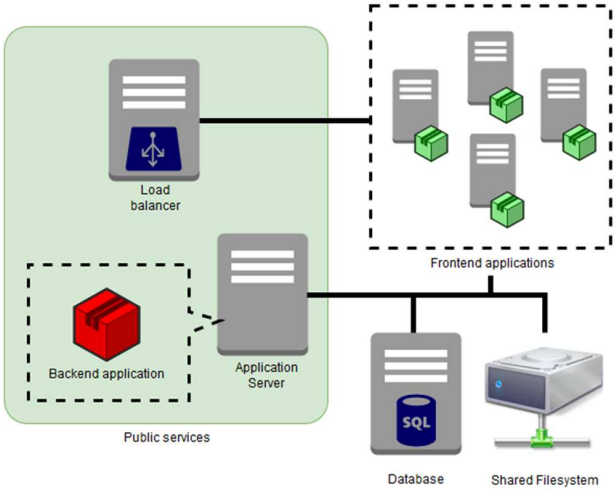

Fig. 6. Replication and Workload Balance.

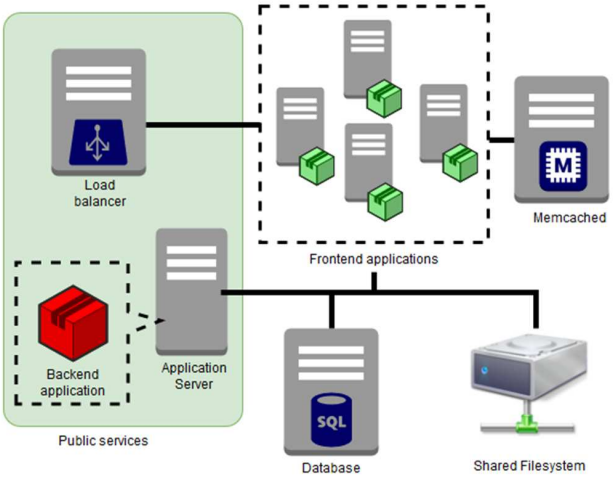

Fig. 7. Session Management with Memcached.

The database could be replicated for data redundancy and availability. Database replication environments usually range from master-slave settings to full-fledged replicated databases running on clusters. Considering security issues, a WAF (Web Application Firewall) such as [39] is usually recommended. Each improvement usually involves at least a) handling the proper resources and b) define the interactions with the rest of the (in-production) system.

III. The Infrastructure Management Problem

Every web application architecture change/improvement implies the corresponding infrastructure change or adaptation, at least in terms of resource allocation. More specifically, the single server configuration shown in Fig. 1 evolved to multiple interconnected servers in a few years. The CMS example we have used as an example (which is described in the previous section) grew from a single server to three servers in less than a year in production. And most of the web applications has grown in some way or another from single to multiple computer servers. Beyond standard and routine tasks for applications and servers such as hardware maintenance, software and data backups, etc. the infrastructure should cope with software updates and new functionalities as produced by the development team/ss. Thus, the operations team/s become more involved with development teams in the daily work.

Virtualization has provided solutions to many well-known infrastructure problems. Replication has been simplified by copying virtual machines, and new servers are installed through graphical interfaces without manipulating physical equipment. In some environments, a backup is *reduced to* snapshot a server's filesystem. However, new problems appear with virtualization. The number of virtual machines that grows rapidly and it is complex to distinguish those relevant virtual machines from those that are not. On the other hand, the replication of virtual machines simplifies a part of the work, but personalizing the new instance continues being a manual work. Hand-made, non-automatic configurations and/or tuning are prone to errors, even following documentation guides. The scenario is even worse in case of disaster recovery, where documentation is hard to find.

Virtualization does not solve the problem of updating an application already in production. Handing source code or software evolution has several well-known and established tools, mostly via SCM (Software configuration Management) tools. Similar tools are not so well-known and established for solving the deployment problem/s. Deploying and updating web systems varies depending on the programming language and the requirements of the application. While software based on Java and .Net require compilation and packing steps, those based in interpreted languages (e.g. PHP, Ruby, Python) do not require them. However, the risk of making a site unstable, unavailable, or faulty is independent of those details. Deployment updates clearly needs to be automated. Furthermore, the deployment should be able to be verified/certified in methodological ways.

Capistrano [3] was one of the first tools for automating update of PHP applications, such as that described in the previous section. Capistrano simplifies deployment and version control using a simple directory structure (releases - current - shared). Even when Capistrano is a step towards deployment automation by means of specific and well-defined conventions, it does not handle or provide aids for problems such as application software dependencies (e.g. libraries) or determining hardware resources, also known as (server) provisioning.

There are several tools for the hardware/server provisioning tasks, and all of them allow to manage the infrastructure as code or IAC [17]. IAC allows to automate the installation of servers, including specific configurations. Products such as Chef [4], Puppet [27], Ansible [1], and SaltStack [34], among others, provide IAC frameworks. Beyond provisioning, IAC allows to apply software development methodologies (e.g. for debugging and testing) to infrastructure management. Chef has been used to configure each of the servers in our CMS (previous section), and it has been also possible to test the infrastructure using KitchenCI [7], ChefSpec [6] and ServerSpec [36]. Then, it is possible to analyze the operation in different platforms like Debian, Ubuntu, CentOS, etc.

Programming the infrastructure with Chef also simplified

handling similar environments for development, testing and QA (Quality Assurance). These different but similarly built environments allow both developers and stakeholders to visualize changes in environments like those in production. Furthermore, Chef provides a detailed infrastructure catalog, whether virtual, physical or in-cloud computers, except those not built/handled by Chef itself. The catalog is indexed using Apache Solr [23], allowing queries about the infrastructure. Even when Chef has proved to be very useful, several complex scenarios may require much time to build and configure. Such scenarios include those with large runtime setups such as non-compatible operating systems and runtime environments (e.g. Symfony 1.0 - PHP 5.x - Ubuntu LTS). Containers have been proposed and are currently used as a more comprehensive solution to several infrastructure problems [21], including the aforementioned one. In particular, Docker [11] is an excellent choice for handling environments as described above. Its growing popularity in heterogeneous and complex environments also offers an option for new developments, narrowing the gap between development and production. Docker simplifies the definition of complex architectures that are (at least partially) shared for development and production or operations (DevOps) environments through Docker Compose [12].

IV. Code Deployment

Code deployment, whether for a new application or an update of an in-production one, can be done manually or using the tools mentioned in the previous section. Table I shows several advantages and disadvantages of the following tools: Capistrano, Chef, and Docker.

TABLE I. Code Deployment Tools

Tool	Pros	Cons
Capistrano	Simple to use, it runs from a terminal and is able to update several sites in parallel from SCM tools.	Shared settings must be manually copied before the first installation or update. Infrastructure or provisioning is not handled.
Chef	Automation from SCM. Installation of services and packages. Infrastructure handling and testing.	Complex, steep learning curve. Several complex environments are hard to handle and test for correctness (e.g. PHP compile in new OS).
Docker	Simple to use: the same configurations in all environments. Allows packaging of legacy applications. Rapid convergence of infrastructure. Simplifies complex architectures.	Conceptual change of handling infrastructure, monitoring, and management of logs and statistics. If used in cluster mode, it requires a container scheduler with complex tuning.

Docker seems to be the "great winner" so far, providing solutions for a problem that Chef handles only partially in complex scenarios. In some way, Docker has capitalized and taken advantage of the previous knowledge about code and infrastructure management (provisioning, testing, etc.) for the problem of deploying software, including the update of an in-production web application. The conceptual change mentioned in Table I above as a Docker disadvantage has introduced some questions about performance, for example, which are now being addressed from several points of view, such as in [14] [31].

A. Continuous Delivery and Continuous Deployment

Continuous delivery and continuous deployment [8] are emerging concepts from agile methodologies used in development. Depending on the application, it is possible that several changes of a software product are generated in a single day. Beyond that, the acceptance of versions must be decided as soon as possible. Thus, the least human intervention and the most automatic testing tend to reduce errors and minimize time-to-market. The sequence for a new software version usually follows several sequential steps: unit, functional, and integration tests, a tagged version is then installed in an automatic testing environment. Once all tests are successful, the application could be made available to the operations area (i.e. continuous delivery) for manual deployment, or automatically installed in production (i.e. continuous deployment).

While SCM tools allow collaboration of development teams through versioning of sources, other important details such as whether to use semantic versioning or not are decided by development teams. The suggestion is to have an explicit correspondence between the version information and SCM tags. Furthermore, the configurations and settings must be clearly documented, at least specifically in development, testing, and production environments. Also, the development flow usually includes or even is based on TDD (Test Driven Development), taking into account passing specific tests before each change or added functionality in the code. Several SCM related tools such as Gitlab, Github, and Bitbucket include code revision step/s as part of the so-called push or merge *code requests*. Associating the code requests with TDD would provide a more automated and methodological way of application update. Another step towards automated continuous delivery or even continuous deployment would be the use of SCM hooks, which are specific actions or tasks triggered at specific points in the SCM execution (e.g. pre- or post-event/s). Thus, developers, testers, and the operation team can set hooks for triggering specific tests. Depending on the tests and automatized complexity, it would be possible to have pre-defined QA rules and policies.

B. Infrastructure Automation

As mentioned before, virtualization plays a very important role in defining (new) infrastructures for software applications. There are numerous virtualization alternatives, the most popular being VMWare VSphere [40], Citrix Xen Server [9], RedHat Virtualization [28], Proxmox Virtual Environment [25], and Microsoft Hyper-V [43], among others. In addition, there are options such as PaaS (Platform as a Service), and IaaS (Infrastructure as a Service) in cloud environments [32]. All of them solve many network and infrastructure issues that must be considered in virtualization environments.

Choosing a virtualization or cloud vendor solution has to be analyzed from taking into account different technical issues: integration with available hardware, storage solutions, networks, application size and dynamic evolution. Also, non-technical issues, such as administrative difficulties to contract cloud services and implementation costs should be considered too. Initially, Proxmox was initially used for virtualization in the web application we presented above, and VMware vSphere was institutionally adopted, and replaced Proxmox. Chef recipes were used for defining the Proxmox infrastructure, so it was rather simple to replace Proxmox by VMware virtualization. However, each server has to be individually identified in the recipes, and it makes almost impossible to automate the complete programming of the infrastructure. At this point, new tools are focused to document, define, and instantiate the complete infrastructure. The virtualization technology can be homogeneous, heterogeneous, and even include/handle nodes in various providers in the cloud. These tools make extensive use of the APIs provided by virtualization software as well as PaaS and IaaS providers.

New and specific tools for provisioning such as Terraform [15] and Chef Provisioning [5] are based on handling IAC at organization data centers. We have used Chef Provisioning to automate the complete assembly and configuration of the VMware vSphere-based infrastructure. We have been able to code the creation, destruction, modification and scaling of the complete infrastructure. And, as noted before, the infrastructure currently handled in VMware can be migrated to another virtualization technology or even manage vendor instances in the cloud with similar (or the same) Chef recipes.

V. Conclusions And Further Work

The web application development process has grown and consolidated in recent years. This growth generates new ways of work and interactions among development and operations teams that traditionally were almost disjoint and independent of each other. Besides, new services and configurations are now needed to be defined at the infrastructure level. And infrastructure definitions should be taken into account not only by the operations team/s but also by source code development team/s. Otherwise, the whole application is in risk of losing functionality, availability, or performance. All of this modifies the traditional organizational structure, where the separation of functions among developers and operators was determined in the context of other paradigms. The new work organizations are based on dynamic and collaborative relationships where the boundaries are not as clear as many years ago.

Continuous delivery and continuous deployment also imply redefining many individual tasks, methodologies, and individual decisions (e.g. on documentation or infrastructure) in terms of the whole process. The high availability as well as in-production application update requirements lead to strong testing of the software as well as the interaction among software and the underlying infrastructure. Virtualization has provided many flexible ways of defining hardware infrastructure environments, but applications usually impose other requirements libraries, middleware, etc. which are beyond hardware. Thus, handling IAC lets to test infrastructure with techniques similar to well-known development methodologies, such as TDD.

The current DevOps tools aid in the continuous delivery and deployment processes, but methodologies have still to be adapted or defined and established. Furthermore, given the current dynamic requirements (e.g. on high availability and in-production application updates) more automatized ways for verification, tests, and QA in general are needed. An (almost, at least) automated integration of tests for source code, current tools for source code management (SCM), and infrastructure as code (IAC) is still to be defined and accepted by institutions and the software industry. The so-called continuous integration pipeline, ranging from source code to in-production application update still has a long way to go.

References

[1] Ansible, Automation for anyone, https://www.ansible.com/ (Retr. 2017).

[2] M. Babar, A. Brown, I. Mistrik, Agile Software Architecture. Aligning Agile Processes and Software Architectures, Elsevier, 2013.

[3] Capistrano, A remote server automation and deployment tool written in Ruby, http://capistranorb.com/ (Retr. 2017).

[4] Chef Software, Inc., Automate IT Infraestructure, (Retr. 2017) https://www.chef.io/chef/.

[5] Chef Software, Inc., Chef Provisioning-Chef Docs, (Retr. 2017) https://docs.chef.io/provisioning.html

[6] Chef Software, Inc., ChefSpec, https://docs.chef.io/chefspec.html (Retr. 2017).

[7] Chef Software, Inc., KitchenCI: Infrastructure Code Deserves Tests Too, http://kitchen.ci/ (Retr. 2017).

[8] L. Chen, Lianping, "Continuous Delivery: Huge Benefits, but Challenges Too". IEEE Software. 32 (2): 50–54, 2015.

[9] Citrix Systems, Inc., Optimized server virtualization for all your data center workloads, https://www.citrix.com/products/xenserver/ (Retr. 2017)

[10] S. Corlosquet, A. Byron, A. Berry, Using Drupal: Choosing and Configuring Modules to Build Dynamic Websites, 3rd Ed., O'Reilly Media, 2017.

[11] Docker Inc., Build, ship and run any app, anywhere, (Retr. 2017) https://www.docker.com/

[12] Docker Inc., Docker Compose: Compose is a tool for defining and running multi-container Docker applications, (Retr. 2017) https://docs.docker.com/compose/

[13] Drupal, Open Source CMS, https://www.drupal.org/ (Retr. 2017).

[14] W. Felter, A. Ferreira, R. Rajamony, J. Rubio, An Updated Performance Comparison of Virtual Machines and Linux Containers, IBM Research Report, 2014.

[15] HashiCorp Terraform, Write, Plan, and Create Infrastructure as Code, https://www.terraform.io/ (Retr. 2017).

[16] P.-H. Kamp, Varnish HTTP Cache, https://varnish-cache.org/ (Retr. 2017).

[17] M. Kief, Infraestructure as Code, O'Reilly Media, Inc., 2015.

[18] G. Kim, J. Humble, P. Debois, J. Willis, The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations, IT Revolution Press, 2016.

[19] K. Król, WordPress Complete, 5th Ed., Packt Publishing, 2017

[20] K. C. Laudon, C. Guercio Traver, E-Commerce 2017, 13th Edition, Pearson, 2017.

[21] Linux Containers, A userspace interface for the Linux kernel containment features, https://linuxcontainers.org/ (Retr. 2017).

[22] Memcached, A distributed memory object caching system: Free & open source, high-performance, distributed memory object caching system, https://memcached.org/ (Retr. 2017).

[23] L. Moczar, Enterprise Lucene and Solr, Addison-Wesley Professional, 2017.

[24] Open Market, FastCGI (1996), Open Market FastCGI: FastCGI

Specification (Retr. 2017). https://fastcgi-archives.github.io/FastCGI_Specification.html

[25] Proxmox Server Solutions GmbH, Open-Source Virtualization Platform, https://www.proxmox.com/en/proxmox-ve (Retr. 2017).

[26] PHP-FPM: A simple and robust FastCGI Process Manager for PHP, https://php-fpm.org/ (Retr. 2017).

[27] Puppet, Open source configuration management tool, (Retr. 2017) https://puppet.com/.

[28] Red Hat Inc., Optimized server virtualization for all your data center workloads, https://www.redhat.com/en/technologies/virtualization, (Retr. 2017).

[29] RedisLabs, Redis, https://redis.io/ (Retr. 2017).

[30] Refinery CMS, Ruby on Rails CMS, http://www.refinerycms.com/ (Retr. 2017).

[31] H. Rosenberg, Performance of Web Applications in Docker Containers on VMware vSphere 6.5, VMware Performance Study, 2017.

[32] N. B. Ruparelia, Cloud Computing, The MIT Press, 2016.

[33] S. Salehi, Mastering Symfony, Packt Publishing, 2016.

[34] SaltStack, Intelligent orchestation for the software-defined data center, https://saltstack.com/ (Retr. 2017).

[35] Sensio Labs, Symfony, High Performance PHP Framework for Web Development, https://symfony.com/legacy (Retr. 2017).

[36] ServerSpec, RSpec tests for your servers configured by CFEngine, Puppet, Ansible, Itamae or anything else, (Retr. 2017), http://serverspec.org/.

[37] S. Souders, S., High Performance Web Sites. Essential Knowledge for Front-End Engineers, O'Reilly Media, 2007.

[38] A. Stellman, J. Greene, Learning Agile: Understanding Scrum, XP, Lean, and Kanban, O'Reilly Media, 2013.

[39] Trustwave Holdings, Inc., ModSecurity: Open Source Web Application Firewall, https://www.modsecurity.org/ (Retr. 2017).

[40] VMware, Inc., Server Virtualization Software | vSphere | VMware, https://www.vmware.com/products/vsphere.html (Retr. 2017).

[41] B. Wilder, Cloud Architecture Patterns, O'Reilly Media, Inc., 2012.

[42] WordPress, Blog Tool, Publishing Platform, and CMS – WordPress, https://wordpress.org/ (Retr. 2017).

[43] L. Yang, D. Zhu, J. Woolsey, S. Rajaram, Achieving over 1-Million IOPS from Hyper-V VMs in a Scale-Out File Server Cluster using Windows Server 2012 R2, Microsoft White Paper, 2014.