LATIN.SCIENCE
XVI Congresso Latino-americano
de Software Livre e Tecnologias Abertas
LATINOWARE 2019

Technical Experiences on a Microservices-oriented Transformation using Open Source Software

Walter Benitez-Davalos, Fabio López-Pires, David Cabañas, Yessica Bogado-Sarubbi
Itaipu Technological Park Hernandarias, Paraguay
{walter.benitez,fabio.lopez,david.cabanas,yessica.sarubbi}@pti.org.py

***Abstract*—Adopting emerging computing paradigms such as cloud applications include challenges associated to transform legacy software to essential characteristics of the mentioned computing model, e.g. on-demand self-service, broad network access, resource pooling, rapid elasticity and measured services. This paper presents a summary of technical experiences on applying one of the most popular approaches to address transformation of legacy software to cloud-native applications: a microservice-oriented architecture with connections to legacy systems through anti-corruption layers. Several technical considerations are presented, focusing on Open Source Software to include particular features on modern development practices as well as solving issues related to the cloud-native transformation.**

***Keywords*-Micro-services; Cloud-Native Applications; Transformation; Modern Software Architecture; Containers.**

I. Introduction

Cloud computing model presents several advantages for modern infrastructure, platform and software, which are provided as services in a pay-as-you-go basis [1]. In this context, software applications following cloud-native design principles and architecture guidelines have inherent advantages in fulfilling current user requirements when executed in complex scheduled environments. These type of applications are commonly known as *Cloud-Native Applications* (CNAs).

Considering the mentioned emerging software paradigm, traditional web-based applications are considered legacy applications, and transforming them to CNAs include several challenges that need to be addressed. For this, systematic methodologies were already proposed [2] including experiences on applications that are still under development [3].

Currently, one of the most popular approaches to address transformation of legacy software to cloud-native applications is to consider a microservice-oriented architecture with connections to legacy systems through anti-corruption layers. Several technical considerations should be discussed to include particular features on modern development practices as well as solving issues related to the cloud-native transformation process, where *Open Source Software* present great alternatives for practical implementations.

This paper presents a summary of technical experiences on an on-going work for applying mentioned approaches for the transformation of a legacy *Enterprise Resource Planning* (ERP) system considering Open Source alternatives on a microservice-oriented architecture.

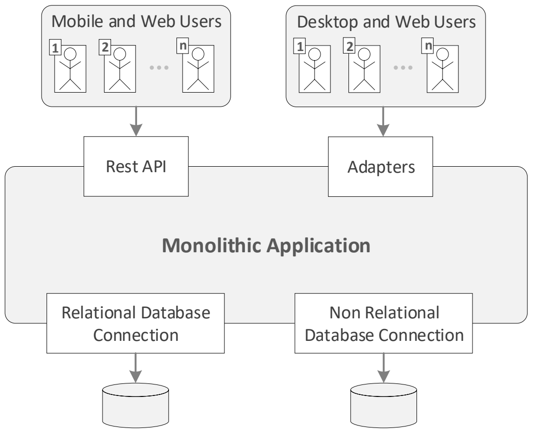

Figure 1. A General Monolithic Software Architecture Diagram.

II. Traditional versus Modern Architectures

In classic software architectures we have top-down dependencies relationships, where software is part of a big block that work with total dependence between modules that interconnects within. External modules and clients connect with the software through *Application Programming Interfaces* (APIs), and adapters and persistence modules exists, traditionally, within the application server. Every part of it is somewhat dependent on other parts. Some degree of modularization is expected, but in any way software is served as just one application. Fig. 1 presents a general diagram of a monolithic application.

On the other hand, in modern software architectures that are cloud-oriented, we have totally independent modules. In this case, software is divided in little pieces, everyone of them with their own domains and data storage. Transactions between modules are managed by patterns like SAGAS and code replication is not much of an issue but is recommended that each module manage different part of the distributed system and just one process. Fig. 2 presents a general diagram of a microservice-oriented application.

The following sub-sections detail relevant aspects related to the legacy ERP system considering a monolithic architecture, as well as transformation approaches for each aspect towards a microservice-oriented architecture.

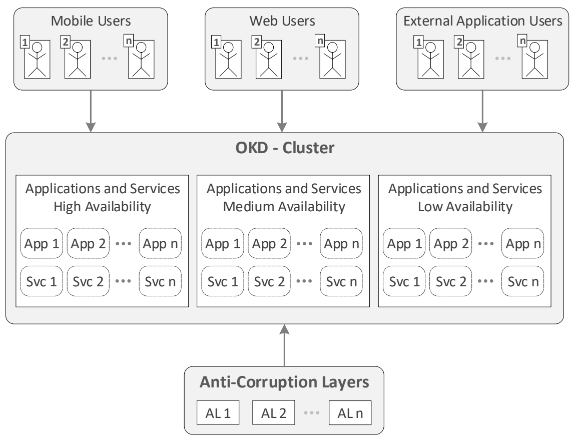
Mobile Users
Web Users
External Application Users
OKD - Cluster
Applications and Services High Availability
Applications and Services Medium Availability
Applications and Services Low Availability
App 1
App 2
App n
Svc 1
Svc 2
Svc n
Anti-Corruption Layers
AL 1
AL 2
AL n

Figure 2. A General Microservice-oriented Architecture Diagram.

A. Development

The development process of the legacy ERP system was based in just one *Source Code Management* (SCM) repository, i.e. Subversion, where each software developer should have deep knowledge of the complete source code to avoid breaking dependencies and crashing the system in production after changes. Essentially, polymorphism and modularization is used for avoiding rewriting code.

Modern development follows Conways Laws [4], and in the proposed microservice-oriented architecture, services are totally independent and connects to functionalities of other groups through APIs. Each group has its development, test and production environments. An agile software development approach is considered for taking relations with final users. It is recommended all functionalities to be stateless and storage should be used as a service consumed by these modules.

B. Deployment

For the deployment of the considered legacy ERP system, packaging was mainly considered. This is automatically obtained through deployments platforms like Jenkins or manually managed by software developers. After this process, infrastructure admins deploy the software in a test environment, mainly as a virtual machine, that is relative similar to the production environment. Software developers test the system manually. Finally the system is deployed in the production environment finishing the, mostly non-automated, software deployment cycle.

In the proposed transformation, and considering that development follows an agile approach, deployment is totally automated as each repository has its own script for their test, development and production environments that automatically triggers when a push is made. Infrastructure admins just manage the automation process, and leave the rest to software developers, adopting this way a DevOps role.

C. Obstacles

The legacy ERP system presents some obstacles, considering a monolithic approach, we have that is slow for iterations between software changes, it has almost non-existent elasticity because of the waste of resource in the escalation of the application, high dependence on developer teams that are working on the software and poor testing that eventually leads to problems in the production environment.

With the proposed transformation, transaction between modules is hard because the complete Atomicity, Consistency, Isolation, Durability (ACID) properties are not achievable. Explicitly, consistency is lost because each module has its own domain. Here, message-brokers and SAGAS [5] patterns could help to resolve this issue, but it is harder to resolve in monolithic applications. Also, documentation should be a must in this type of architecture, because of the independence between groups of work.

III. Transformation Experiences

As previously mentioned, technical experiences of the transformation of a legacy ERP system into a cloud-native one is summarized in this work. In the following subsections, technical aspects related to infrastructure, as well as software development aspects are presented, including main considerations. Additionally, Table I summarizes the proposed transformation open source software tools with the corresponding advantages on implementing them.

First technical decisions include issues related to the original legacy infrastructure, taking into account that was too complex (and large) to be totally migrated in a short period of time. Additionally, there were so many dependencies to handle and it was currently being frequently used. Then a migration of the old infrastructure was totally out of question.

Secondly, there were budget limitation to work with private software. This was not a very difficult issue to solve, considering the broad alternatives of open-source platforms with the appropriate features to work in the transformation. To adopt the proposed changes, we should analyze how the original infrastructure works and what is the correct strategy to deal with an hybrid architecture that brings the best of a microservice-oriented one but without the hustles of changing the original legacy one.

A. Infrastructure

The legacy ERP system considers a pure monolithic web application wrote in Java, served in a Wildfly 10 application server with a Postgresql 8 Database System. The connection to the database was via configuration files in the application server, and the application was deployed within a war packaging format. Changes were done manually on each deployment and tests were executed in development environments. Security was handled through security layers divided by VLANs, where different layers divide networks and each

Development Aspects	In Legacy ERP	In Transformation	
	Legacy Tools	Proposed Tools	Advantages
Source Code Management	Subversion	git	Centralized vs. Decentralized allows developers to work offline and then merge in the external repository.
Code Analysis	Non existent	Sonarqube	Added a measurement of quality in the code.
Continuous Integration	Non existent	Use of pipeline with GitLab Runner	Faster integration between modules and testing.
Continuous Deployment	Non existent	Use of GitLab Runner and OKD templates	Faster iterations and deployments to production.
Deployment	Manual in VM	Automatic in pods and containers in the OKD clusters	Faster deployment and monitoring is reassured with health checks in containers.
Architecture	Monolithic	Hybrid (Microservices + Legacy Systems)	More robustness to the system, in the sense that each module is domain independent and is assumed a domain driven design.

Table I
SUMMARY OF TRANSFORMATION ASPECTS AND PROPOSED OPEN SOURCE TOOLS.

virtual machine connects to each other. The infrastructure was handled by KVM virtualization running in a Peacemaker cluster of physical machines.

The proposed infrastructure is deployed over the old one, i.e. the peacemaker cluster will be used to deploy virtual machines that will be the core of the application platform. In this case, the technology used will be Openshift Origin or OKD as is currently called. OKD is an open source platform given by the company Red Hat that works on top of Kubernetes and priorities security in containerized applications as it is made for the industry. Internally OKD will manage its own Software-defined Networks (SDNs), where it will automatically deploy PODs in different nodes as Kubernetes. These nodes are:

- **Master Node**: The master node is the host of hosts, it will manage nodes in its Kubernetes cluster and schedules pods to run on nodes.
- **Infrastructure Node**:Is a node that runs the infrastructure services and allows a high availability cluster if the compute nodes are down.
- **Compute Node**: The compute node is the one that host the applications and handles all the computation load.

Connectivity to the legacy ERP system is done via what is currently called anti-corruption layers. The idea of this is to create a legacy domain-specific application that communicates with new applications. This could be done via a facade inside the application itself or in a total different application that connects directly with the database or with some communication systems of the application.

The deployment of the platform is made via virtualization in bare metals, it is used peacemaker to create a cluster of machines that could deploy the VMs used to host the platform. This is made to facilitate the migration to another infrastructure if needed, as shown in Fig. 3.

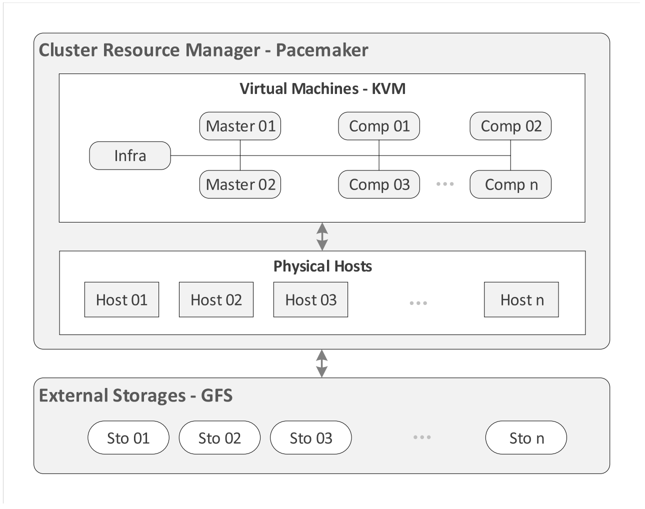

Figure 3. Legacy and Proposed Infrastructure.

B. *Continuous Integration and Deployment (CI/CD)*

Source Code Management (SCM), and version control, is proposed to be done via git repositories, considering three branches of code (i.e. dev, test and master). On each push operation, to any of these branches, automated tests and deployment is activated. For this, gitlab, gitlab-runner and pipelines are used, as described in Table I. For efficiency reasons, a code analyzer called Sonarqube is used and gives a report on each push operation of the dev branch as presented in Fig. 4.

IV. CONCLUSION AND FUTURE DIRECTIONS

Main identified conclusions and future directions to continue research of the present work are summarized next.

In this on-going work, we presented an initial transformation of a set of legacy software with a monolithic architecture to a hybrid legacy + microservice-oriented architecture, mainly using a set of open source tools and platforms.

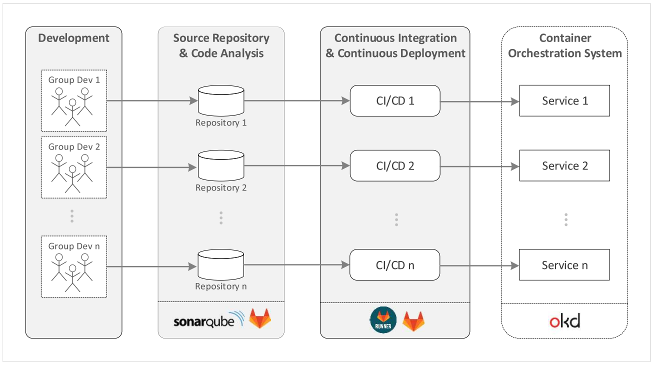

Figure 4. CI/CD pipeline process for the proposed transformation.

As we saw in Section I, the use of decentralized code management as git, a code analyzer as Sonarqube and decentralized platforms to deploy PODs and containers in a secure way as in OKD provide us with the enough flexibility to integrate legacy software architectures without removing them and also give us the tool to make a gentle migration without hurting the software operation. Additionally, in a direct association with the Conways Law, we could deliver software in a total independent way as we grow our team.

Future work will also include a serverless platform for serving and deploying functions into our ecosystem, as well as data-oriented applications, analytic mobile applications and others, that would feed a future data-warehouse.

Additionally, the presented transformation process would open some fields on possible research as:

- extend the transformation to Platform as a Service (PaaS) service model,
- include an experimental evaluation for integration of PaaS service providers [6],

Finally, several future directions may include formal systematic definitions of a general-purpose (no application-specific) methodology for transformation of legacy software architecture into modern ones, such as those based on micro-services or serverless architectures.

- problems associated with continuous use of serverless functions in an infrastructure,and how to plan accordingly when to use pods and/or containers instead of them,
- how to implement a Function Hub into our current transformation ecosystem [7],
- how to define an optimal granularity of services or what measures we can take on each service [8].

References

[1] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities," in *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on*. Ieee, 2008, pp. 5–13.

[2] G. Toffetti, S. Brunner, M. Blöchlinger, J. Spillner, and T. M. Bohnert, "Self-managing cloud applications: design, implementation, and experience," *Future Generation Computer Systems*, vol. 72, pp. 165–179, July 2017.

[3] J. Spillner, Y. Bogado, W. Benítez, and F. López-Pires, "Co-transformation to cloud-native applications: development experiences and experimental evaluation," in *8th International Conference on Cloud Computing and Services Science (CLOSER), 19-21 March 2018, Funchal, Madeira*. Scitepress, 2018.

[4] J. D. Herbsleb and R. E. Grinter, "Splitting the organization and integrating the code: Conway's law revisited," in *Proceedings of the 1999 International Conference on Software Engineering (IEEE Cat. No. 99CB37002)*. IEEE, 1999, pp. 85–95.

[5] C. Richardson, "Microservices pattern: Sagas." [Online]. Available: https://microservices.io/patterns/data/saga.html

[6] J. M. Pintos, C. N. Castillo, and F. López-Pires, "Evaluation and comparison framework for platform as a service providers," in *2016 XLII Latin American Computing Conference (CLEI)*. IEEE, 2016, pp. 1–11.

[7] Y. Bogado-Sarubbi, W. Benitez-Davalos, J. Spillner, and F. Lopez-Pires, "Towards sustainable ecosystems for cloud functions," in *ESSCA, Zurich, Switzerland, December 21, 2018*. CEUR-WS, 2019, pp. 18–24.

[8] M. R. López and J. Spillner, "Towards quantifiable boundaries for elastic horizontal scaling of microservices," in *Companion Proceedings of the10th International Conference on Utility and Cloud Computing*. ACM, 2017, pp. 35–40.