# The Challenges of Developing a Multi-Domain Microservices Platform: The Case of REQSTER

Robert Ramač[1,2] and Vladimir Mandić[1,2]
[1] TIAC doo, Ćirila i Metodija 13a, Novi Sad, Serbia
[2] University of Novi Sad, Faculty of Technical Sciences, Trg Dositeja Obradovića 6, Novi Sad, Serbia
ramac.robert@uns.ac.rs, vladman@uns.ac.rs

**Abstract**

As software trends go, in recent years, cloud technologies have become increasingly popular. Thanks to cloud technologies, nowadays software solutions can be developed and deployed in significantly shorter time cycles. These solutions usually rely on some services to do the heavy lifting and processing of the data. At the other hand, microservices emerged as an attractive technology for implementing backend services. The advocates of the domain driven design (DDD) acknowledge the microservices as the technology for constructing DDD solutions. At our practice, we recognized the potential to use microservices for domain specific projects using DDD principles. Furthermore, we see future opportunities and benefits of a multi-domain microservices platform. In the past year, we started with the development of REQSTER, a microservice platform that is used as a backend solution for the development of apps for different business domains. In this paper and talk we will present two main groups of challenges encountered during the initial development of REQSTER: (1) operational complexity, and (2) using DDD for developing a multi-domain platform.

## 1 Introduction

TIAC doo (www.tiacgroup.com) is a software development company founded in 2004 in Novi Sad, Serbia that focuses on the latest technologies, high development standards and good internal processes. In past years we have dealt with over 200 projects that vary in technology, domain and industry. All of TIACs client projects were domain specific and their development was approached from a domain specific perspective. Over the years we noticed the following patterns: common and reusable design, the need for mobile apps and a shorter time to market. To every client the application is a set of features and these features tend to overlap for some applications while there will always be some that are different. It is these similarities that are targeted by our multi-domain platform.

In order to meet our clients needs, we set our business goals to speed up the development process, increase the delivery rate of software products and new functionalities, optimize the testing process and increase the quality. In the past year, we started the development of REQSTER[1], a platform that offers a complete palette of backend services. For clients REQSTER represents a set of cloud services accessible through standard *restfull* APIs. While, for us REQSTER represents a set of procedures and rules on how to develop various microservices. In what follows we present an overview of the platform and two main groups of challenges that we encountered during the initial stages of development.

# 2 Overview of the REQSTER platform

This section gives an overview of the REQSTER platform. First we will go over some basic technical details concerning the main components of the platform. Afterwards, we will discuss the deployment chain and present its main stages.

## 2.1 Technical overview

Figure 1 depicts the main components of the REQSTER platform and their interaction with each other. The top level components are the services, support services, admin and system dashboard.

Each service consists of multiple layers. First is the token based security that will be explained a bit later because it requires mentioning of component interaction. The WebAPI layer is REST based and used to enable client-service communication. The feature negotiation protocol is reserved for handling the negotiation of features. The util layer consist of multiple sublayers like event notifications, logging and error handling. In the end the database (DB) access layer is used for data persistence, if needed.

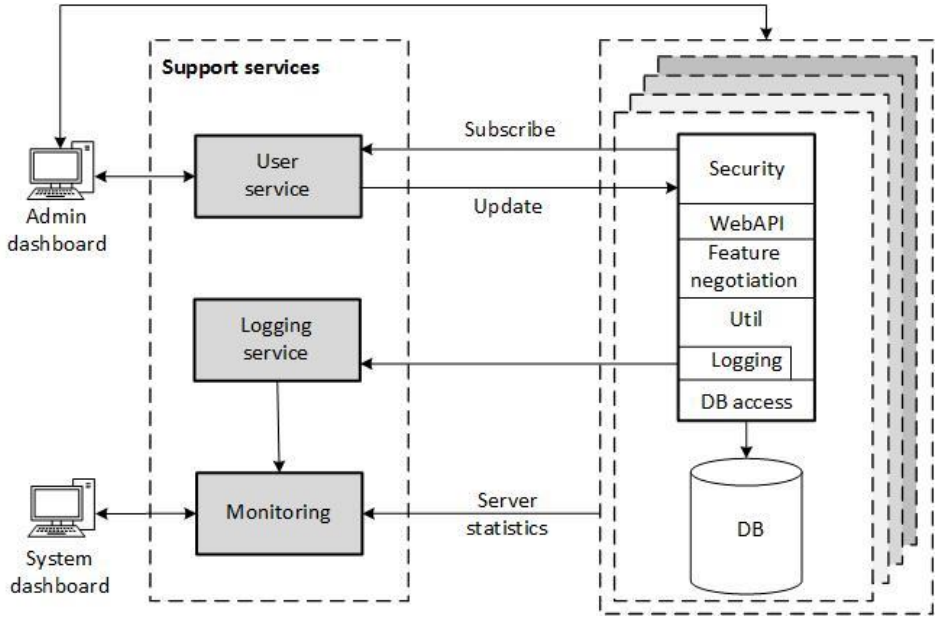


**Figure 1** – Technical overview of the REQSTER platform

[1] REQSTER homepage - http://www.reqster.com/ (visited 08.31.2017.)

The admin dashboard handles the administration of all services using REST calls to services endpoints. The system dashboard is used to monitor the status and performance of all services.

The logging service is used to log data from each of the services, while the monitoring service uses this data and also collects the data from servers and makes it available to the system dashboard.

The security mechanism is implemented in the following way. Whenever a new instance of a service is set up the service subscribes to the user service. In this way the service lets the user service now of its existence and the user service logs the information. In return the user service updates the newly registered service by giving it the list of all logged users and their privileges. If the newly registered service is stopped for some reason when it comes online again it will simply reregister. On the other hand, if the user service is stopped the first call to the user service from any service will have to be authenticated and then the service is again added to the user service list of subscribed services.

## 2.2 REQSTER deployment chain

Figure 2 depicts the REQSTER deployment chain. The first two stages of the deployment chain are concerned with maintaining the code base for each service and producing builds that are automatically tested. The rest of the deployment chain is dealing with the deployment of the instances of the services.

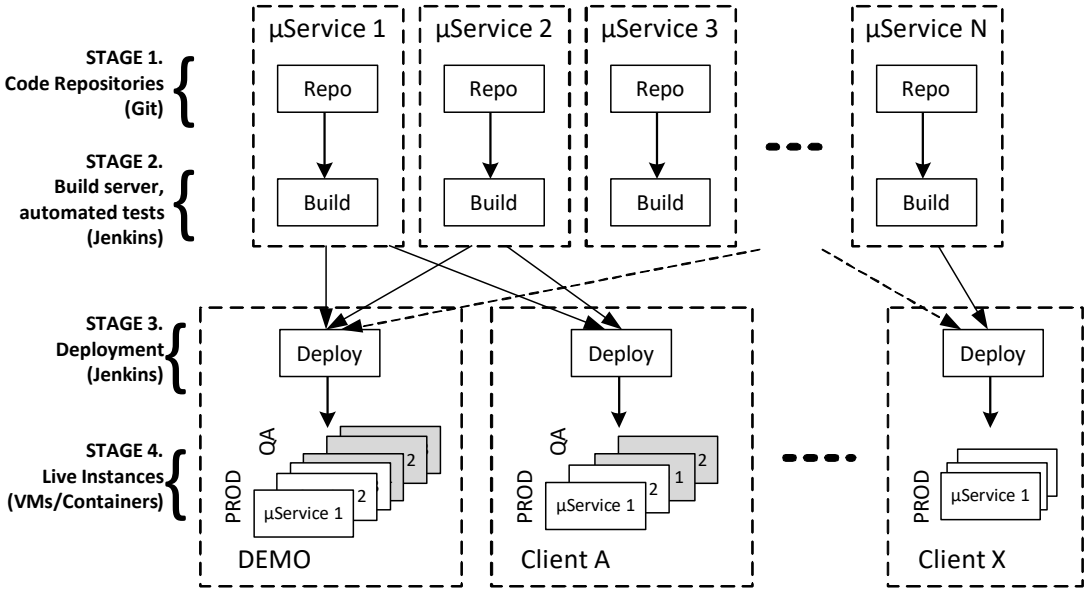


**Figure 2** – REQSTER chain of deployment

Deployment chain starts when a developer commits code to a repository (Stage 1). Each service has its own independent repository, after committing the code the second stage, i.e. build and test process, commences. A build job is triggered by new commits into the repo. Currently, Jenkis [4] is used for the build server. Each service is companioned with a set of unit tests, which ensure that all implemented features (also from previous version) are working correctly. After the first stage before the build process, it is possible to include additional quality assurance activities that are based on static code analysis, like SonarQube [3].

One of the specifics of the REQSTER platform is that it allows to clients run dedicated instances of the services. Therefore, the deployment process is client oriented. In stage 3, the deployment starts by combining the latest versions of the services for the specific client. Instances of the services are

deployed at virtual machines or in containers. As an option, clients can separate testing (QA) instances from the production (PROD) ones.

# 3 The challenges

Communication is crucial for a success of software projects, it spans across many different channels: in-between development team, between teams and customers, among customers or users and so on. The importance of communication is accentuated in DDD through a principle of *ubiquitous language* [1]. The language that is used in a targeted domain reflects on and permeates software design and source code. However, similar or same domains do not necessarily share the same language, e.g. in one context, an order represents a task to be implemented, and while in another context, a service to be performed. Language is one of the key differentiators of *bounded contexts* [1]. Features are specified in this language and need to be translated into service functionality. The main motivator for developing the microservicies platform is to be able to deploy autonomous services that can be reused on different projects [2]. Certainly, the ability to use existing services is beneficial for quicker development and delivery, higher quality (due to reuse of live and tested code), and better team organization.

At the other hand, we encountered some strains with the development and implementation of the REQSTER platform. The two main groups of challenges were: (1) Operational complexity and (2) Utilizing DDD design for a multi-domain platform.

The first group of issues concerns the technology itself and includes the complexity of development as well as the deployment of the microservices platform. During the development and deployment process there is a need for supporting services and scripts for automating the deployment process.

In an ideal scenario, the deployment process is triggered by a commit to the microserivice's repository, and automatically all stages of the deployment chain should be successively initiated. However, in REQSTER, the deployment jobs (Stage 3, Figure 2) are initiated semi-automatically, i.e. they are not triggered by the new coming build from Stage 2 (Figure 2), but rather initiated periodically every 24h, with an option to set different deployment periods for different clients. The additional complexity to the deployment process is that different clients can have different sets of service instances. The whole deployment process is managed by a configuration file committed to the services' repositories, in that configuration file, among other things are listed active clients for whom the service in question needs to be deployed.

The second main issue was concerned with the impact on the DDD concept. As mentioned, in the case of REQSTER the main idea was achieving reusability of the platform on various projects. This directly comes into conflict with the idea of domain driven design and building very specific solutions.

# 4 Discussion

In this talk we will present and discuss how we managed the operational complexity, mainly by developing supporting services and tools, like a system dashboard and utility services capable of analyzing log data for easier performance monitoring and debugging.

As for, designing services that can be used in multiple domains, we see it as an open issue that calls for continuous design reevaluations. However, we noticed that the implementation of feature negotiation protocols could decrease the need for designing new services, by allowing different users from different domains to use same services with different sets of features. Further benefits of the feature negotiation protocol is the reduced number of instances of services, specifically eliminating the need for QA instances of services.

# References


[1] Evans, E. (2004). *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional.

[2] Newman, S. (2015). *Building microservices: designing fine-grained systems*. O'Reilly Media, Inc.

[3] Campbell, G., & Papapetrou, P. P. (2013). SonarQube in action. Manning Publications Co.

[4] Smart, J. F. (2011). Jenkins: The Definitive Guide: Continuous Integration for the Masses. O'Reilly Media, Inc.