Migrating Monolithic Applications to Microservices-based Customizable Multi-tenant Applications

Sindre Grønstøl Haugeland[1], Phu H. Nguyen[2], Franck Chauvel[2], and Hui Song[2]

[1] University of Oslo, Oslo, Norway
sindrgro@ifi.uio.no
[2] SINTEF, Oslo, Norway
firstname.lastname@sintef.no

Abstract

Software providers have a myriad of legacy monolithic single-tenant software running on client infrastructure. Migrating legacy software to a multi-tenant cloud solution requires a structured and planned approach, in a context with no defined best practices on how to achieve this. This paper suggests an approach for how software providers can migrate legacy software to the cloud, providing customers with a flexible customization possibility, while taking advantage of the economics of scale that the cloud and multi-tenancy provide.

1 Our Migration Approach

Our approach focuses on migrating applications that follow the MVC design pattern. It draws inspiration from the migration approach proposed by E.Wolff [8] in a survey about migration approaches, and the generic re-engineering tool in [4]. Modifying the Blueprint approach to include the initial phase of the proposed tool by Kazman allows the migrating party to gain an understanding of the current application before starting the migration. The approach becomes a three-phased approach, where the initial phase consists of analyzing the application and discovering bounded contexts and domains in the application according to domain-driven principles [1]. The next two phases occur in parallel, where we extract functionality from the existing application while building the necessary infrastructure to support the new services.

To test the approach, we applied it to the SportStore application [2]. The application following the MVC pattern [2] is a web-based store for sports equipment. SportStore is implemented in .NET Core with Views, Models, and Controllers for ordering, product catalogs, and a session-based shopping cart. We use these "groupings" as our bounded contexts during the analysis and extract functionality from them during the decomposition part of the migration. After this, we start implementing services to cover the functionality of the existing application and set up the infrastructure to support it. The infrastructure includes typical components like the API-gateway and a form of back-end communication for the services. Figure 1 shows the target architecture of the SportStore application that we have used for our migration experiments.

2 Multi-tenancy and Deep Customization

Our approach aim at enabling the target architecture to be customizable for multi-tenant context as presented in [5, 6]. The approaches in [5, 6] offer tenants a way to (deeply) customize the functionality of the multi-tenant application without interfering with behavior for other tenants. The customization-driven aspect make our approach different from other migration approaches like [7]. First, we focus on introducing multi-tenancy to the application. To support multi-tenancy, we need a system for Identity Access Management (IAM), and to support

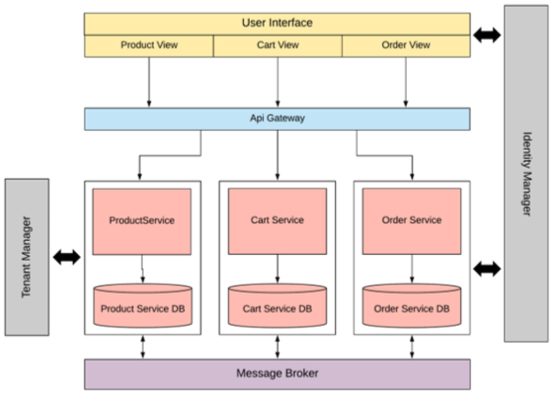

Figure 1: The target architecture with customization possibility in a multi-tenant context

customization of the application for the tenants and to configure the storage to isolate tenant data, we need a tenant manager. The tenant manager provides all the registered customizations and endpoints for the logged-in user, which is retrieved using a bearer token issued by the IAM system. Tenant isolation at application level is crucial to avoid data leaks problem between tenants as raised in [3]. With the tenant manager and the IAM system in place, we start adding support for customization. We use the tenant manager to return external endpoints to customized functionality in cooperation with the IAM system to ascertain the "tenantID" of the user. The main service then reroutes the request to the external endpoint along with the information required by the customized function.

References

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

[2] Adam Freeman. *Pro Asp. net Core Mvc*. Apress, 2016.

[3] Andrei Furda, Colin Fidge, Alistair Barros, and Olaf Zimmermann. Chapter 13 - reengineering data-centric information systems for the cloud – a method and architectural patterns promoting multitenancy. In Ivan Mistrik, Rami Bahsoon, Nour Ali, Maritta Heisel, and Bruce Maxim, editors, *Software Architecture for Big Data and the Cloud*, pages 227 – 251. Morgan Kaufmann, 2017.

[4] R. Kazman, S. G. Woods, and S. J. Carriere. Requirements for integrating software architecture and reengineering models: Corum ii. In *Proceedings Fifth Working Conference on Reverse Engineering (Cat. No.98TB100261)*, pages 154–163, 1998.

[5] Phu H. Nguyen, Hui Song, Franck Chauvel, Roy Muller, Seref Boyar, and Erik Levin. Using microservices for non-intrusive customization of multi-tenant saas. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, page 905–915, New York, NY, USA, 2019.

[6] Hui Song, Phu H Nguyen, and Franck Chauvel. Using microservices to customize multi-tenant saas: From intrusive to non-intrusive. In *Joint Post-proceedings of the First and Second International Conference on Microservices (Microservices 2017/2019)*, 2020.

[7] Davide Taibi and Kari Systä. From monolithic systems to microservices: a decomposition framework based on process mining. In *8th International Conference on Cloud Computing and Services Science, CLOSER*, 2019.

[8] Eberhard Wolff. Migrating monoliths to microservices: A survey of approaches. 2019.