Experience Report: First Steps towards a Microservice Architecture for Virtual Power Plants in the Energy Sector

Manuel Wickert[1], Sven Liebehentze[1], and Prof. Dr. Albert Zündorf[2]

[1] Fraunhofer IEE
manuel.wickert@iee.fraunhofer.de sven.liebehentze@iee.fraunhofer.de
[2] University of Kassel, zuendorf@uni-kassel.de

Abstract

Virtual Power Plants (VPPs) provide energy sector stakeholders with a useful abstraction for decentralized energy resources (DERs) by aggregating them. Supporting software systems operate as a part of the critical infrastructure and must handle a fast-growing number of DERs. Modern architectures like Microservice architectures can therefore be a good choice for dealing with such scalable systems where changing market and regulation requirements are part of daily business. In this report, we outline our first experiences changing from an existing VPP software monolith to a Microservice architecture.

1 Introduction

In today's green energy transition, the primary concept of a modern power plant is beginning to become that of VPPs consisting of a huge number of small DERs [3]. Usually these DERs are wind farms, photovoltaic parks, biogas plants, energy storage or flexible loads. Since VPPs are the modern kind of a big power plant, they are replacing conventional large power plants over time.

In several research project, Fraunhofer IEE has developed and evaluated such a software system, the VPP software solution IEE.vpp, which is in operation for some utilities and trading companies in the energy sector. The software became a huge monolith over time with many hundred KLOC (kilo lines of code). It consists of a canonical data model that was good in some situations but also very inflexible in others. In situations where the data model did not fit to the corresponding business logic, the development time for the components increased. In the energy domain, where the regulatory framework changes very often and new business models must be implemented quickly, this will become an issue over time. Furthermore, it was hard for domain experts with algorithmic skills to contribute to the VPP solution because their main programming languages are Python and Matlab, but the VPP solution was written in Java. We therefore decided to migrate our macro architecture to a Microservice [2] approach based on a Domain-Driven Design [1] and implement some Microservices in Python instead of using only Java.

In this report, we present the first steps of our migration towards this Microservice architecture, done stepwise within our agile development process to keep the provision of regular updates to our customers ongoing.

2 Architectural Migration

First, we analyzed the business model of our users and identified the main use cases to support the business functions. We determined the following four use cases:

- Administration of DER

- Monitoring and Control
- Flexibility Optimization
- Information Provision

We approached this cases by case, starting with the flexibility optimization first, because the domain model for this component was already very different from the canonical data model and the component was not so extensive in comparison to the other components. Moreover, the experts for optimization at our institute are not experienced in the current programming language of the IEE.vpp software Java, and cannot then easily contribute their knowledge. We decided then to implement the new service in Python to integrate our optimization experts in the future development team.

Looking at the old monolithic architecture, the optimization component was implemented as a high-level layer based on the aggregation, persistence and interface layers (including basic validations) (see 1. in figure 1). This was the starting point of our migration.

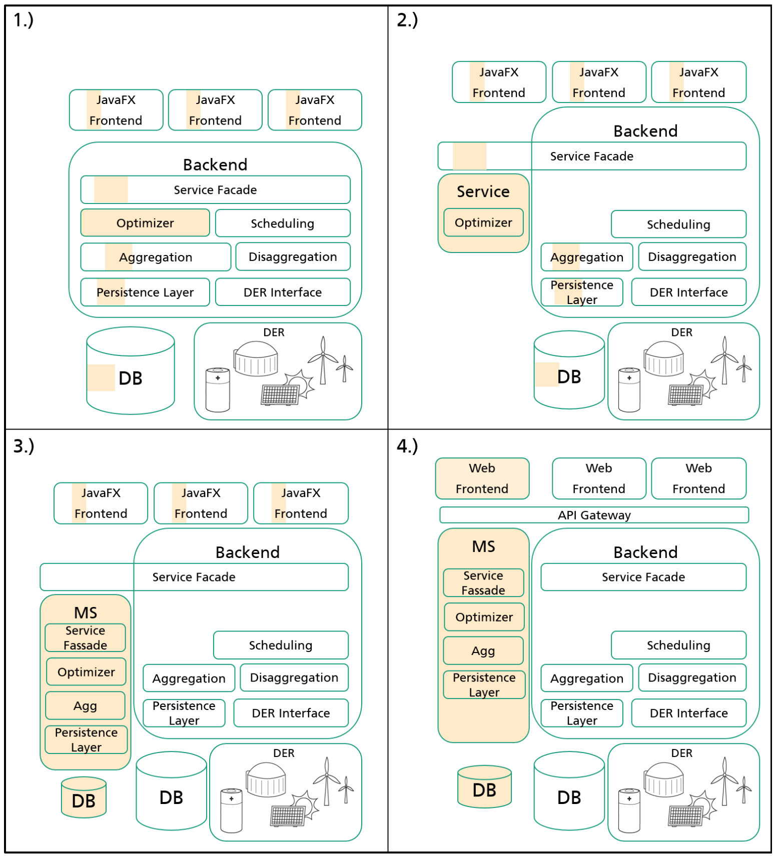

Figure 1: Migration Steps

The migration can be divided into four main steps that are shown in figure 1:

1. In the first step we identified the business logic and parts in the surrounding components

that are relevant to processing input and output data for the logic. Exemplary parts of panel 1 (figure 1) are marked yellow.

2. For the second step, we extracted only the business logic in a SOA-like service and called these service directly from the layer where it was extracted from. Here, the data model in the SOA Service is changed and we used an Anti-Corruption Layer to transform the information between the monolith and the new service. The other layers, e.g. persistence and aggregation, as well as the UI, stay in the monolith. After this step, all existing integration and contract tests from the monolith could be used to verify that VPP software works as before.

3. In step three, we enhanced the SOA-Service to a Microservice and shifted the relevant parts of the aggregation and persistence layer from the monolith while adding a new database to the service as well. The existing service facade served as an API Gateway.

4. The last step is going to build a micro front-end and use an external API Gateway. The full migration should then be done and independent deployment possible.

The migration steps were all executed during our SCRUM-based development process. After each of these steps we could perform existing contract tests and partial integration tests and deploy our software in production. That made us more flexible to implement new features in parallel.

3 Conclusion

With our stepwise approach, we were able to provide continuously working software within our agile development, even without a pure Microservice approach during the transition. Currently, we mainly benefit from the inclusion of further expert knowledge from the development team by using Python as the programming language of the new Microservice. Also, introducing bounded contexts with an anti-corruption-layer [1] in order to separate the systems has made extending the optimization model much easier than before. The next step is to evolve the other use cases into Microservices using the described migration path.

We have shown that the approach worked for migrating a VPP software system from the energy domain. This approach should however be applicable in any domain with a need to extract part of a business logic into a Microservice in another programming language. The important aspects consider analyzing the software based on domain knowledge, extracting only the business logic, proceeding with the existing infrastructure from the monolith in step two, then integrating persistence and finally taking idenpendent UI considerations.

References

[1] Evans. *Domain-Driven Design: Tacking Complexity In the Heart of Software.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[2] Martin Fowler and James Lewis. Microservices. 2014.

[3] H. Saboori, M. Mohammadi, and R. Taghe. Virtual power plant (vpp), definition, concept, components and types. In *Proceedings of the 2011 Asia-Pacific Power and Energy Engineering Conference*, APPEEC '11, pages 1–4, Washington, DC, USA, 2011. IEEE Computer Society.