Shaping & Building your initial architecture

case studie

S67-3

2021

Inhoud

[The case. 2](#_Toc66362458)

[Onderzoeksvragen 3](#_Toc66362459)

[Strategie 3](#_Toc66362460)

[Resultaten 5](#_Toc66362461)

[Monolithisch naar macroservices naar microservices 5](#_Toc66362462)

[Monolithisch naar microservices 7](#_Toc66362463)

[Monolithische modules 7](#_Toc66362464)

[Conclusie 8](#_Toc66362465)

[Bronnen 9](#_Toc66362466)

# The case.

“Author: Patrick de Beer

Date: December 8, 2020

The customer relations department (CRD) of the Dutch Railways (Nederlandse Spoorwegen, NS) is responsible for all contact between customers and the NS and the sale of train tickets, including international tickets. In addition, they take care of refunds for delays or when a customer forgets to check out. Also, they manage the distribution of the magazine ‘Spoor’ which is sent to all subscribers 4 times a year.

In the near future, CRD likes to expand their services to provide a full travel experience for their customers, including the reservation of train tickets, hotel rooms, bicycles, etc., both with national and international destinations. They also want to explore other features together with a NS customer panel.

The director of the Dutch Railways believes that, in the long run, it should be possible to travel by train without a ticket, i.e., using face recognition or another type of biometric authentication.

The CTO of the Dutch Railways thinks that everything will move to the cloud. He has heard that serverless is the future. Offered services should be reliable and performance should automatically scale when requests for specific services increase. It should be possible to develop services and manage services runtime independently.

An *event storming* session has taken place to hear all the stakeholders' needs and create a common understanding of the domain and how to move forward.

At this moment there is an existing software system with a technical heritage that is not near future proof. The organization currently depends on a monolithic system built in Java EE7 with all customer-related data stored in a single database. The application runs on servers owned by the NS. A backup system is in place. Although the system meets the requirements at present, it is foreseen that expanding the software to support the new services may become a challenge. In particular, when biometric data needs to be stored. In addition, software engineers having experience with Java EE7 are becoming scarce. The principal software architect has decided to migrate the current system to a microservice architecture to prepare for future expansion and to get experience with cloud technology.

Could you advise the principal architect on the *process* of moving from the current monolithic application to an architectural style that is s*calable and maintainable* and can meetup to the future plans and non-functional requirements, like e.g. offering to large number of simultaneous service requests? There is a not negotiable requirement that the introduction of the new platform can be done gradual: phasing out old functionality and introducing new services gradually, without major interruptions of the system.”

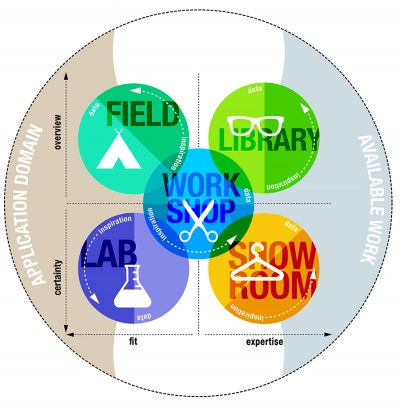
# Onderzoeksvragen

Het probleem dat in deze case studie beschreven wordt valt samen te vatten met de onderstaande onderzoeksvraag.

*“Hoe kan een monolithische applicatie geleidelijk gemigreerd worden naar een microservice omgeving zodat deze applicatie schaalbaar en onderhoudbaar blijft?”*

# Strategie

Het verrichten van het onderzoek zal gebeuren met behulp van het DOT Research framework. Voor het gebruik van het framework is het van belang dat er verschillende methodes worden gebruikt. Deze methodes komen uit verschillende strategieën en worden toegepast om je onderzoek te valideren.



Om de onderzoeksvraag te beantwoorden hebben wij gekozen voor de volgende strategieën met daarbij de volgende methodes.

|  |  |  |
| --- | --- | --- |
| Field | Library | Workshop |
| Focus group | Community research | Decomposition |
|  | Literature study | IT architecture sketching |

Wij het zijn onderzoek begonnen met een combinatie van ‘Literature study’ en ‘Community research’. Dit is een goed startpunt voor het onderzoek omdat veel andere bedrijven al voor hetzelfde probleem hebben gestaan. Daardoor zijn er veel guides en artikelen te vinden over de stappen die bij het proces horen en de best practices daarvoor.

Vervolgens hebben we de strategie ‘Focus group’ binnen onze projectgroep toegepast. Onze groep heeft vanaf semester 3 al met microservices gewerkt, waardoor iedereen voldoende kennis heeft om een goede bijdrage aan de discussie te leveren.

De strategieën ‘Decomposition’ en ‘IT architecture sketching’ kunnen binnen dit onderzoek zelf niet ideaal gebruikt worden. Deze twee zijn echter wel erg belangrijk wanneer iemand de opdracht beschreven in de case studie aan zou nemen.  
Ondanks dat het om een monolithisch systeem gaat, komt ‘Decomposition’ van pas omdat het migreren geleidelijk moet gebeuren. Door het opdelen van het probleem in kleinere stukken kan het oude systeem feature voor feature gemigreerd worden. Daarnaast is ‘IT architecture sketching’ erg belangrijk. Door stap voor stap de nieuwe services in kaart te brengen naast de huidige architectuur verloopt de migratie erg soepel.

# Resultaten

## Monolithisch naar macroservices naar microservices

Het eerste resultaat wat we hebben gevonden is het omzetten van een monolithische applicatie naar een macroservice en deze weer omzetten naar een microservice. Een macroservice heeft het voordeel dat hij nog met dezelfde datastore verbonden kan zijn als de monoliet. Daarnaast is het mogelijk om meerdere dataobjecten te hebben binnen de macroservice. Dit zorgt ervoor dat objecten die moeilijk te scheiden zijn in het monoliet makkelijker te migreren zijn naar een microservice oplossing.

Het is mogelijk dat macroservices nog steeds dezelfde last van schaalbaarheid en onderhoud hebben als een monoliet. Maar omdat ze gefocust zijn op het verminderen van objecten en interacties, is het veiliger deze tussenstap te maken dan in één keer naar microservices te gaan. Dit zorgt er ook voor dat het niet ten koste gaat van de uptime.

Door van monoliet naar macroservices te gaan creëer je ook een beeld van hoe de services verder opgesplitst kunnen worden naar microservices.

Om soepel van een monolithische applicatie naar een microservice te gaan zijn de volgende stappen uitgestippeld:

1. Identify logical components  
   Het is van belang om eerst de logische componenten van de applicatie te schetsen om zo een beeld te vormen van wat er speelt. Een belangrijk punt is om te kijken of verschillende applicaties dezelfde soort data output hebben, en of deze hierom samengevoegd kunnen worden.
2. Flatten and refactor components  
   Nu alles geïdentificeerd en gegroepeerd is kan er georganiseerd worden binnen de groepen. Het is vooral te kijken naar componenten met vergelijkbare functionaliteit. Er is echter bij microservice maar één service die de functionaliteit uitvoert.
3. Identify component dependencies  
   Nadat alles is gereorganiseerd is het belangrijk om te kijken welke componenten afhankelijk zijn van elkaar. Dit kan gevisualiseerd worden met tools zoals bijvoorbeeld Lucidchart.
4. Identify component groups  
   Nadat de dependencies duidelijk zijn, kan er gefocust worden op de component groepen binnen de applicatie. Deze zullen vervolgens omgezet worden naar microservices wanneer dit in één keer mogelijk is. Of eerst naar een macroservice omgezet worden. Het doel is om een kleine set objecten hun acties logisch te separeren voor het uiteindelijke systeem.
5. Create an API for remote user interface  
   De remote user interface is bedoeld als communicatie tussen de gebruikers en het systeem. De interface moet te gebruiken zijn tijdens de migratie maar ook daarna. Het doel van deze migratie is om een universele API te creëren waar de user en de applicatie data kan manipuleren. Vrijwel alles hangt samen rond deze API, dus zal het moeten ontworpen worden zodat het de bestaande interacties niet in de weg liggen. Daarnaast moet het ook de optie bieden om er makkelijk nieuwe toe te voegen.
6. Migrate component groups to macroservices (move component groups to separate projects and make separate deployments)  
   Het overkoepelende doel van deze stap is het separeren van component groepen en het apart deployen van deze groepen.
7. Migrate macroservices to microservices  
   Nadat de macroservices zijn gemaakt is het makkelijker om te zien hoe de componenten nog verder gesepareerd kunnen worden. Dat wordt in deze stap gedaan.
8. Deployment and testing.  
   Nu de services op zijn plek staan is het mogelijk om ze te testen. Dit kan gedaan worden in de vorm van end-to-end en intergration tests. Wanneer de tests aangeven dat het systeem geen gebruik meer maakt van de bestaande datastore, kan er naar productie gepushed worden.

## Monolithisch naar microservices

De tweede optie om geleidelijk over te gaan naar een microservice architectuur is het direct overzetten van functies naar microservices, in plaats van de tussenstap naar macroservices te maken, hoe dit in elkaar steekt is als volgt.

Er wordt eerst nagedacht welke stukken code hun eigen microservice krijgen, als dit gedaan is wordt er voor dit stuk code een microservice geschreven. De volgende stap is dan om deze code uit de monolithische applicatie te verwijderen en in plaats daarvan een API call te maken naar de microservice. Als alle verschillende microservices uiteindelijk zijn gemaakt en gekoppeld aan de monolithische applicatie, hou je een soort van gateway over. Dit zorgt ervoor dat nog een architecturaal probleem opgelost wordt zonder veel extra werk (terwijl er natuurlijk wel een stukje extra werk moet worden gedaan om het daadwerkelijk een goed werkende gateway te maken).

Onze conclusie is dan ook dat dit een goede manier zou zijn, en dat we het kunnen afwegen tegen de andere mogelijkheden die wij in dit document aanbieden.

## Monolithische modules

De derde optie voor het migreren naar een microservice structuur beloopt de minst steile helling. Dit proces bestaat uit een aantal stappen welke hieronder kort beschreven worden:

1. split de code,
2. split de databases,
3. definieer een standalone service,
4. gebruikt de standalone service,
5. verwijder de functionaliteit uit de monoliet.

Door code die met verschillende datasets in aanraking komt in de monolithische applicatie al te splitsen in verschillende modules maakt dit de migratie naar microservices later veel eenvoudiger. De tweede stap is het loskoppelen van de database. Hierdoor is de module in de monoliet eigenlijk niet meer afhankelijk van de rest van de monoliet. Een logische vervolgstap is dan ook het schrijven van een microservice die de functionaliteit van de monoliet overneemt. Zodra deze microservice volledig werkt, kan deze gebruikt worden door de monoliet en kan de code in de monoliet verwijderd worden.

Doordat dit proces heel geleidelijk loopt is dit een erg geschikte methode voor de migratie die in de casestudie wordt beschreven.

# Conclusie

Na drie verschillende manieren van migreren naar microservice architectuur te hebben bekeken, zijn we tot een centrale conclusie gekomen. Deze is als volgt:

Om ervoor te zorgen dat het systeem wordt overgezet zonder grote storingen te krijgen in het proces, bevelen wij aan om een combinatie van monolithische modules en macroservices te gebruiken. In de praktijk betekent dit dat er volgens het monolithische modules systeem wordt gewerkt. Maar als het blijkt dat stukken code te sterk gekoppeld zijn, dan is er een optie om hier eerst een macroservice van te maken en deze later om te zetten naar microservices.

Wij hebben uiteindelijk voor deze strategie gekozen omdat terugkijkend naar het onderzoek, dit de snelste en meest geschikte optie blijkt te zijn voor de casus. Tot slot is er op deze manier meer dan genoeg mogelijkheid om de applicatie uit te breiden en makkelijk te onderhouden met de hedendaagse technieken en technologieën voor bijvoorbeeld hosting en CI/CD. En dus voldoet de strategie aan alle requirements die opgezet zijn voor de casus.

# Bronnen

Frye, B. (2020, 28 sept). *8 Steps for Migrating Existing Applications to Microservices*. Geraadpleegd op 11 maart 2021 van:  
<https://insights.sei.cmu.edu/sei_blog/2020/09/8-steps-for-migrating-existing-applications-to-microservices.html>

Dehghani, Z. (2018, 24 apr). *How to break a Monolith into Microservices*. Geraadpleegd op 11 maart 2021 van:  
<https://martinfowler.com/articles/break-monolith-into-microservices.html>

Richardson, C. (2020). *Refactoring a monolith to microservices*. Geraadpleegd op 11 maart 2021 van:  
<https://microservices.io/refactoring/>