Microservices and Event Storming Report

Project: Microservices and Event Storming

Project team: BBB-BC

Team: Joeri Berman (2834499)

Luc Urlings (3071081)

Stijn Baltessen (3064565)

Wouter Vandenboorn (3099571)

Floris Feddema (3188256)

Davy de Haas (3099695)

Opdrachtgever: Bartosz Paszkowski

Version: 0.2

Version Date: 13 maart 2020

Status: Concept

# Inleiding

In dit document zal de casus over de implementatie van microservices voor de Nederlandse Spoorwegen worden behandeld. Er zal een advies worden geschreven over hoe de transitie van het oude monolithic systeem naar de nieuwe microservices implementatie het beste kan worden uitgevoerd.

We beschrijven verder hoe de transitie naar het nieuwe systeem uitgevoerd kan worden zonder dat er services tijdelijk niet beschikbaar zijn. Hierbij houden we ook rekening met de mogelijke transitie naar een (serverless) cloud omgeving.

Ook geven we informatie of de principes van microservices die moeten worden toegepast.

# Inhoudsopgave

[**Inleiding**](#_npg4ywfxrn6q) **2**

[**Inhoudsopgave**](#_fpxodst0eum) **3**

[**Microservices principes**](#_s8a8xknorqbe) **4**

[**Aanpak**](#_22zgdshn2dv2) **4**

[**Bespreek de case**](#_xwqib6o4lbxi) **5**

[**Identificeer de vragen**](#_7e7t0tp9w4wg) **5**

[**Brainstorm**](#_trrw5cdzhl1a) **6**

[**Analyseer en structureer**](#_8w2rdv21w97l) **8**

[**Formuleer doelen**](#_adsrzn2e1cb2) **8**

[**Studeer individueel**](#_v89gyfh314ps) **9**

[**Bespreek bevindingen**](#_l9ps9cx7u622) **11**

[**Advies**](#_s365sqz9fcqa) **13**

# Microservices principes

1. Zorg ervoor dat iedere service verantwoordelijk is voor zijn eigen data persistentie. Dit is een principe van microservices.
2. Iedere service heeft een API zodat andere services hem kan benaderen. Deze stap forceert je na te denken over hoe iedere service met elkaar moet communiceren en welke technologieën hierbij te pas komen.
3. Een API is de enige manier voor het benaderen van service.
4. Het is noodzaak dat een service een eigen codebase heeft en technology stack zodat deze door een eigen development team kan worden ondersteund.
5. Bij microservices staat decentraliseren centraal. Doordat iedere microservice zijn eigen technology stack heeft en het onderdeel zo klein mogelijk is, is het gemakkelijk bij te houden voor een development team.
6. Één microservice hoort bij één business goal.

# Aanpak

## Bespreek de case

Iedereen heeft de casus doorgenomen, hierna hebben we met zijn alle de casus besproken. De uitkomsten hiervan zijn:

* Er zijn Microservices nodig.
* De oude omgeving is niet meer onderhoudbaar aangezien de kennis van Java EE 7 nog maar bij weinig developers aanwezig is.
* De opdrachtgever heeft geen goed beeld van Serverless.
* Er zal een migratie naar een Cloud Provider moeten plaatsvinden.
* De bestaande omgeving moet weg van de huidige monolithic applicatie.
* De overgang van het oude naar het nieuwe systeem moet geleidelijk gebeuren en moet een 24/7 uptime behouden.
* Er moet mogelijkheid zijn om nieuwe features op latere momenten gemakkelijk toe te voegen aan de applicatie.
* event storming kan gebruikt worden om te brainstormen over de casus.

## Identificeer de vragen

Tijdens het bespreken besloten we welke vragen relevant zijn om te vragen. Deze vragen staan hier genoteerd:

* Wat is event storming? En hoe kan dit gebruikt worden?
* Wat is serverless? En is het in te zetten voor deze casus?
* Hoe wordt een monolithic applicatie naar een microservice based applicatie omgezet? En hoe kunnen we hier 24/7 uptime garanderen?
* Wat zijn de hoofdprincipes van microservices?
* Wat is de cloud en wat heeft dit te maken met serverless en 24/7 uptime?

## Brainstorm

Wat weet de groep al over bepaalde dingen en identificeer potentiële oplossingen.

### Event storming

Voorkennis is er bij maar een iemand uit de groep, die heeft dit voorgelegd bij de rest en de basisprincipes hiervan uitgelegd tijdens deze brainstorm. hieruit kwam de conclusie dat event storming een middel kan zijn om services te identificeren.

### Microservices

Wat zijn de hoofdprincipes van microservices?

* Schaalbaar
* Elk onderdeel eigen data persistence
* Distributed, niet allemaal op 1 systeem
* Modulair
* Single responsibility per service
* Elke service kan andere technologie zijn
* Communicatie tussen services door middel van API endpoints
* Cloud based systemen
* Geen single point of failure
* Makkelijker in onderhoud
* Moeilijk in opzet
* Kleinere teams per service/ het verdelen van ontwikkeling in kleinere teams

### Serverless

* Cloud providers
* Amazon Lambda, Azure functions
* Betalen per verzoek
* Niet nadenken over de structuur om de applicatie heen
* Makkelijk op te zetten
* Afhankelijkheid van de hosting provider
* Uitleg medestudent: het individueel hosten van endpoint(s), javascript.

### Cloud technologieën

* Kubernetes
* Rancher
* Docker
* AWS, Azure, Google Cloud
* Schaalbare hosting
* 99.99% SLA(Service level agreement)/uptime
* Cloud opslag
* Gelimiteerd aan netwerk capaciteiten (Latency, bandbreedte)
* Databases
* Niet in eigen beheer
* Linux/Windows Server
* Machine learning
* GPU renderfarms
* Deployment in verschillende regio’s
* Content Delivery Network

### Transitie van monolithische software naar microservices

* Groot proces
* Geleidelijk
* Uptime risico's
* Monolitisch: uitleg, een grote applicatie zonder single responsibility. moeilijk onderhoudbaar door schaal.
* Rekening houden met transitie data op databases

## Analyseer en structureer

Door deze brainstormsessie hebben we onze kennis met elkaar kunnen delen. Hierdoor beschikt iedereen in het team over dezelfde basiskennis over: event storming, serverless, cloud technologieën en de transitie van monolitische software naar microservices.

## Formuleer doelen

Doelen opzetten om de kennis van onbekende punten van de case te achterhalen.

De vragen die nog onbeantwoord of boven water zijn gekomen na de brainstormsessie zijn:

* Is serverless toe te passen op onze casus?
* Hoe kunnen we hier 24/7 uptime garanderen tijdens de transitie van monolithische applicatie naar micro services?
* Wat is event storming?

## Studeer individueel

Probeer de kennis te achterhalen die in hiervoor zijn opgesteld. Zet hier de activiteiten in per student wat er gedaan is om de doelen te behalen.

### Davy

**Event Storming**

* [Modelling Reactive Systems with Event Storming and Domain-Driven Design](https://blog.redelastic.com/corporate-arts-crafts-modelling-reactive-systems-with-event-storming-73c6236f5dd7)
* [How to use event storming to achieve domain-driven design](https://techbeacon.com/devops/introduction-event-storming-easy-way-achieve-domain-driven-design)

**Is serverless toe te passen op onze casus?**

Ik heb de casus gelezen en kwam tot de conclusie dat deze casus niet de benodigde informatie heeft om requirements op te kunnen stellen.

### Floris

<https://martinfowler.com/articles/break-monolith-into-microservices.html>

<https://medium.com/@springdo/a-facilitators-recipe-for-event-storming-941dcb38db0d>

### Joeri

**Event storming**

* <https://techbeacon.com/devops/introduction-event-storming-easy-way-achieve-domain-driven-design>
* <https://www.lucidchart.com/blog/ddd-event-storming>

**Serverless**

* <https://www.cloudflare.com/learning/serverless/what-is-serverless/>
* <https://cloud.google.com/serverless>
* <https://aws.amazon.com/serverless/>

### Wouter

* Van Monolith naar microservices.  
  <https://www.nginx.com/blog/refactoring-a-monolith-into-microservices/>  
  Advies, stop met het uitbreiden van de monolith applicatie. het wordt zo alleen maar erger. Maak een plan voor de implementatie met microservices, en begin met het verwerken van losse microservices.
* <https://medium.com/@walkingtreetech/strategies-to-migrate-monolithic-application-to-microservices-b2028eb4873b>

### Stijn

* Event storming: [Modelling Reactive Systems with Event Storming and Domain-Driven Design](https://blog.redelastic.com/corporate-arts-crafts-modelling-reactive-systems-with-event-storming-73c6236f5dd7)
* serverless casus: meer informatie over casus nodig
* Hoe kunnen we hier 24/7 uptime garanderen tijdens de transitie van monolithische applicatie naar micro services?: <https://martinfowler.com/articles/break-monolith-into-microservices.html>

### Luc

*Is serverless toe te passen op onze casus?*

<https://serverless-stack.com/chapters/what-is-serverless.html>

<https://hackernoon.com/what-is-serverless-architecture-what-are-its-pros-and-cons-cc4b804022e9>

*Hoe kunnen we hier 24/7 uptime garanderen tijdens de transitie van monolithische applicatie naar microservices?*

<https://martinfowler.com/articles/break-monolith-into-microservices.html>

*Wat is event storming precies?*

<https://www.eventstorming.com/>

<https://en.wikipedia.org/wiki/Event_storming>

<https://www.lucidchart.com/blog/ddd-event-storming>

## Bespreek bevindingen

Iedereen brengt punten van zijn individuele study bij elkaar.

### Event storming

Is een manier van designen van een system. De basis waarop je dit doet is wel anders dan andere manieren.

Je designed dit vanuit een event, dit zorgt ervoor dat je vanuit een een andere kijkpunt doet kijken en een andere gedachtegang krijgt. Event storming doe je samen met een team. Wanneer iemand een vraag heeft over een event zet hij dit event op het bord.

Het gebruikt verschillende kleuren sticky notes om dit te gebruiken.

Er zijn 3 verschillende sticky notes

* Events Rood
* Externe systemen Paars
* Commando’s Blauw

Met Event Storming krijg je een beter beeld van de flow van een applicatie, dit kan voor microservices heel fijn zijn omdat je dan ziet hoe de communicatie tussen de verschillende services gaat.

### Serverless computing

Serverless computing is een uitvoeringsmodel waarbij de cloudprovider (AWS, Azure of Google Cloud) verantwoordelijk is voor het uitvoeren van een stuk code en de resources dynamisch toe te wijzen. Hierbij worden alleen de kosten in rekening gebracht voor de middelen die gebruikt worden om de code uit te voeren.

De code wordt uitgevoerd in stateless containers. Deze kunnen bereikt worden via een aantal mogelijkheden (denk aan http requests, database events, etc). De code die naar de cloud provider wordt verstuurd is meestal in de vorm van een functie. Daarom wordt er ook wel naar gerefereerd als *“Functions as a Service”* oftewel *“FaaS”*.

Doordat de cloud provider deze service al heeft opgezet hoeft de gebruiker niet na te denken over de onderliggende infrastructuur. Bij normale cloud computing is het zo dat je betaald voor de gehele server. Als je daar niet alle resources voor gebruikt betaal je net zoveel als dat je dat wel zou doen.

### 

### Transitie van monolithische software naar microservices

Door van te voren een goede architectuur te maken voor een microservice based systeem is het mogelijk deze verandering zonder downtime uit te voeren.

Er zijn een verschillende manieren waarop dit kan worden gedaan. Door stap voor stap de services om te zetten naar een microservice kan dit op een gecontroleerde manier gebeuren.

Zodra de microservice stabiel is en goed functioneert kan deze uit de monolithische applicatie worden gehaald en gebruik maken van de microservice.

Wanneer alle functionaliteiten op deze manier vervangen zijn door microservices is de hele applicatie gemigreerd.

# Advies

1. Identificeer en bepaal welke microservices er moeten komen en hoe deze met elkaar moeten communiceren. Gebruik hiervoor Event Storming.
2. Hiervoor moet ook een architectuurdocument worden gemaakt, hierbij moet rekening worden gehouden met microservices principes. Voor meer informatie ga naar het hoofdstuk microservices principes.
3. Vind een platform dat het deployen, monitoren en het schalen van services ondersteund. Hierbij is de gegarandeerde uptime een belangrijke requirement om naar te kijken. Denk hierbij aan cloud providers zoals AWS, Microsoft Azure en Google Cloud, deze garanderen een uptime van 99.99%.
4. Per service de service omzetten naar de cloud omgeving. Als deze stabiel is en draait kan deze functionaliteit uit de monolithic worden gehaald en gebruik maken van de microservice. Op deze manier blijft het gehele product altijd beschikbaar en zal de transitie naar een cloud based systeem geleidelijk gaan.
5. Nadat alle services omgezet zijn draait de volledige applicaties in microservices in de cloud.normaal gezien.