Implementering af Microservices med Messaging og integration til eksterne systemer

Af Mads Mikel Rasmussen  
32. januar 2021

Antal tegn: 19200, 8 ns.

Indholdsfortegnelse

[1 Indledning 3](#_Toc61890246)

[2 Problemformulering 3](#_Toc61890247)

[3 Teori 3](#_Toc61890248)

[4 Introduktion til systemet 4](#_Toc61890249)

[5 Anvendt teori 5](#_Toc61890250)

[5.1 REST web API 5](#_Toc61890251)

[5.2 Messaging 5](#_Toc61890252)

[5.3 Microservices 5](#_Toc61890253)

[6 Implementering 6](#_Toc61890254)

[6.1 Service Orienteret Arkitektur som Microservices 6](#_Toc61890255)

[6.2 Kald til eksterne web API 6](#_Toc61890256)

[6.3 Messaging mellem microservices med gRPC 6](#_Toc61890257)

[6.4 Message Content Filtre 6](#_Toc61890258)

[6.5 Message Translators 6](#_Toc61890259)

[6.6 Message Aggregator 6](#_Toc61890260)

[6.7 Push notifikation til klient med SignalR 6](#_Toc61890261)

[7 Konklusion 7](#_Toc61890262)

[8 Perspektivering 7](#_Toc61890263)

[9 Referencer 7](#_Toc61890264)

# Indledning

Der er valgt at konstruere et nyt system med det formål at belyse min opnåede viden, færdigheder og kompetencer i kursets læringsmål. Til dette formål er der lavet følgende case:

Systemet skal hente data fra Kystdirektoratet og Dansk Meteorologisk Institut og notificere brugere om potentielt høj vandstand og pålandsvind ved de danske kyster, da netop disse to faktorer kombineret ofte resulterer i oversvømmelser

Rapporten fortsættes herfra med en problemformulering og den teori der er anvendt til belysning heraf, samt til systemets konstruktion. Casens implementering illustreres ved kodeeksempler, der anvender et udvalgt teoretisk grundlag, i konteksten af problemstillingen.

Slutteligt konkluderes på problemstillingens, samt perspektiveres til de aspekter af kursets læringsmål der gør sig gældende for den anvendte teori i konteksten af det konstruerede system.

# Problemformulering

Casen anvendes til illustration af én potentiel løsning på følgende problemstilling:

*Hvordan konstrueres bedst et nyt system,  
der understøtter messaging i en moderne service orienteret arkitektur?*

Denne problemstilling rejser følgende spørgsmål:

* Hvilken moderne arkitektur bør anvendes?
* Hvilke moderne teknologier bør anvendes til sikring af skalabilitet og agilitet i udviklingsprocessen?
* Hvilke metoder, mønstre og teknikker til systemintegration bør anvendes til casen?
* Hvilke standarder bør overvejes som dataformat, således en langtidsholdbar løsning opnås?
* Hvilke forretningsmæssige overvejelser bør drøftes ved etableringen af et nyt system, ud fra et systemintegrationsmæssigt synspunkt?

# Teori

Jeg redegør her for, hvordan jeg ser Microservices som en moderne variant af SOA.

Et system i en Service Orienteret Arkitektur (SOA), skal ifølge Thomas Erl, opfylde følgende otte designmæssige principper *(Erl, 2008)*:

* Standardiserede service kontrakter: Formålet og funktionaliteten af en service bør eksponeres til klienter på en standardiseret måde, herunder definitionen af datamodeller og datatyper, til sikring af styrbarheden, pålideligheden og konsistensen af servicens endpoint.
* Løs kobling: Både internt i en service og eksternt til andre komponenter, herunder andre services, bør afhængigheder være så begrænset som muligt, med det formål at opnå så stor uafhængighed som muligt i implementeringen, hvilket giver større fleksibilitet i udviklingen og stabilitet i driften.
* Abstraktion: Detaljer og information i implementeringen bør eksponeres i mindst mulige omfang, hvilket afspejles i sammensætningen med f.eks. andre services. Begrebet er meget lig det vi kender fra objekt orienteret programmering.
* Genbrugelighed: En service bør kunne opfylde det samme formål, men på differentieret vis, eksempelvis ved parametrisering, og herved kunne genbruges.
* Autonomi: En service bør kunne opfylde sit formål, i så stor grad af uafhængighed som muligt, af både andre services og driftsmiljøet. Dette fordrer at en service har en vis grad af styring med driftsmiljøet.
* Tilstandsløshed: En service bør eksponere mindst mulig grad af sin logiske tilstand til klienter, hvorved skalabilitet og tilgængelighed fordres. Dette begreb minder meget om det vi kender fra objekters tilstand i objekt orienteret programmering.
* Opdagelighed: En service bør af hensyn til både den tekniske og videnmæssige tilgængelighed, men også til den forretningsmæssige forståelse, være let at identificere som et IT aktiv, med de kapaciteter den indeholder.
* Sammensættelighed: En service bør kunne deltage i forskellige sammensætninger af services på forskellige tidspunkter, der tilsammen understøtter en positiv forretningsmæssig værdi.

I konteksten af moderne distribuerede systemer og systemintegration, er det naturligt at redegøre for hvordan Microservices bør anskues i konteksten af SOA, og særligt Erls otte principper. Martin Fowler argumenter, at SOA *begrebet*, med tiden er for omfavnende og diffust *(Fowler, 2014)*, hvor han eksemplificerer SOA med, at en Enterprise Service Bus, løser f.eks. problemet med integration af mange monolitiske applikationers kommunikation, og skaber herved forretningsmæssig værdi. Men Fowler mener ikke dette er ikke formålet med Microservices. I Microservices sammensættes forskellige services til at løse forskellige udfordringer, og skaber herved forretningsmæssig værdi. Den forretningsmæssige fordel ved microservices over traditionel SOA, er at en forretningsmæssig udfordring muligvis allerede kan løses, helt uden at udvikle ny funktionalitet, da kombinationen af eksisterende funktionalitet således allerede kan muliggøre en løsning på den pågældende udfordring. Microsofts officielle holdning er, citat *(de la Torre, Cesar; Wagner, Bill; Rousos, Mike;, 2020)*:

»SOA was an overused term and has meant different things to different people. But as a common denominator, SOA means that you structure your application by decomposing it into multiple services that can be classified as different types like subsystems or tiers. «

Både Fowler og Microsoft, beskriver microservices, som *”en server applikation som et antal små services”* *(de la Torre, Cesar; Wagner, Bill; Rousos, Mike;, 2020)*. Med denne viden, kan man således argumentere for, at en microservice arkitektur opfylder Erls otte principper for en SOA. Dette kan omsættes i følgende UML 2.5 Component diagram:

# Implementering

På baggrund af ovenstående teori, er der til casen valgt følgende arkitektur:

Hver applikation er skrevet i .NET 5.0 med C# 9.0. Kommunikationen mellem microservice applikationerne foregår med RPC i en message channel, og den konkrete teknologi gRPC er valgt. En klient kan være alt fra et website til en mobiltelefon, dog kræves det at klienten understøtter SignalR til den dynamiske routing fra server til klient. Data hentes fra to datakilder i henholdsvis CSV og JSON format. Her følger en ganske kort beskrivelse af de anvendte teknologier:

* .NET 5.0/C#: Afviklingsplatform til driftsmiljøet, hvor v. 5.0 er en fortsættelse af .NET Core 3.1, og ikke .NET Framework 4.8, som udfases. Microsoft har tilkendegivet, at de satser på nye versioner af .NET hvert år. Dette gør teknologien attraktiv, særligt i Danmark.
* gRPC med protobuf-net:
* SignalR:

med en extension, der muliggør en WCF kontraktbaseret tilgang, i stedet for standard protos i gRPC.

# Anvendt teori

## REST web API

## Messaging

## Microservices

# Implementering

## Service Orienteret Arkitektur som Microservices

## Kald til eksterne web API

## Messaging mellem microservices med gRPC

## Message Content Filtre

## Message Translators

## Message Aggregator

## Push notifikation til klient med SignalR

# Konklusion

# Perspektivering

# Referencer

de la Torre, Cesar; Wagner, Bill; Rousos, Mike;. (2020). Hentet 22. 01 2021 fra dotnet.microsoft.com: https://dotnet.microsoft.com/download/e-book/microservices-architecture/pdf

Erl, T. (2008). *SOA: Principles of Service Design.* Pearson Education, Inc.

Fowler, M. (25. 03 2014). Hentet 18. 01 2021 fra martinfowler.com: https://martinfowler.com/articles/microservices.html#MicroservicesAndSoa