UCL ERHVERVSAKADEMI OG
PROFESSIONSHØJSKOLE

SYNOPSIS

.NET Core og Microservices

Koordinering af forretningsprocesser på tværs af services

Julian Mathias Kock

Underviser
Kaj BROMOSE

20. maj 2020

Indhold

1 Indledning

Vores verden er blevet en digital verden. Alt lige fra ens køleskab og elkedler til valutaer og telefoner er blevet digitale. Internetforbindelserne bevæger sig nu med lysetshastighed og en efterspørgelse på ingen nedetid, hurtige svartider og overskuelighed er blevet hverdag for udviklere.

Softwaren der udvikles har aldrig været under så stort et pres. Udviklere bliver sat til at løse endnu mere komplekse forretningsprocesser hvor hver enkelt kodeblok skal være let læselig, skalerbart og omskiftelig. Samtidig skal koden være gennemtestet, virke i alle mulige forskellige miljøer og integrere let mod andet software.

Med andre ord er arkitekturen samt infrastrukturen blevet vigtigere end aldrig før. For at imødekomme alle disse krav har man kigget mod opdeling af kode i en service baseret arkitektur, hvor hver service tildeles et ansvar og kan blive udviklet og skaleret individuelt. Fokus i dette fag har været Microservices som er et term for mange autonome services der arbejder sammen for at udarbejde og give en forretningsmæssigværdi[1].

Denne opgave vil fokusere på den koordinering der skal til for at Microservices kan løse forretningsmæssigeopgaver, samtidig med at services skal være autonome og opnå eventual consistency.

2 Problemformulering

Hvordan opnår man en fornuftig koordinering af Microservices til at løse forretningsmæssigeopgaver?

Del spørgsmål

1 Hvad er SAGA-mønstret?

2 Hvordan kan man sikre autonome Microservices?

3 Hvad skal din Microservices tage højde for at opnå eventual consistency?

3 1. delspørgsmål

3.1 Saga pattern

Når der tales om koordineringen af Microservices menes der den interne kommunikation/dataudveksling der skal til for at services kan udføre en forretningsproces. Et eksempel på en forretningsproces kunne f.eks. være at lave en ordre der tilsidst afsendes og modtages af en kunde. I en traditionel arkitektur kunne man bruge ACID transaktioner, men når der er tale om flere services er det svært at lave isoleret transaktioner. En måde hvorpå man kan imødekomme dette er ved brug af distrubureret transaktioner, så som 2PC (two-phase commit). Denne løsning kan være tilstrækkelig, men den giver dog nogle ekstra problemer. Ideén med distrubureret transaktioner er at alle services der indgår enten committer transaktioner eller laver en rollback. Det betyder altså at du nu er direkte afhængig af, om alle services er klar til og modtage requestet, eller ej. Derfor vil negativ downtime for en service have effekt på alle andre services der indgår i den distribureret transaktion[2]. Dette bryder på autonom princippet bag Microservices, og der er nu en direkte afhængighed videre fra service til service.

For at imødekomme kommunikation der skal gå på tværs af services kan man her bruge et Saga pattern. Saga mønstret er et mønstre hvorpå man kan lave transak-

tioner på tværs af services. Der findes to typer af sager. Choreography baseret og orchestration baseret[2].

3.2 Orchestration saga

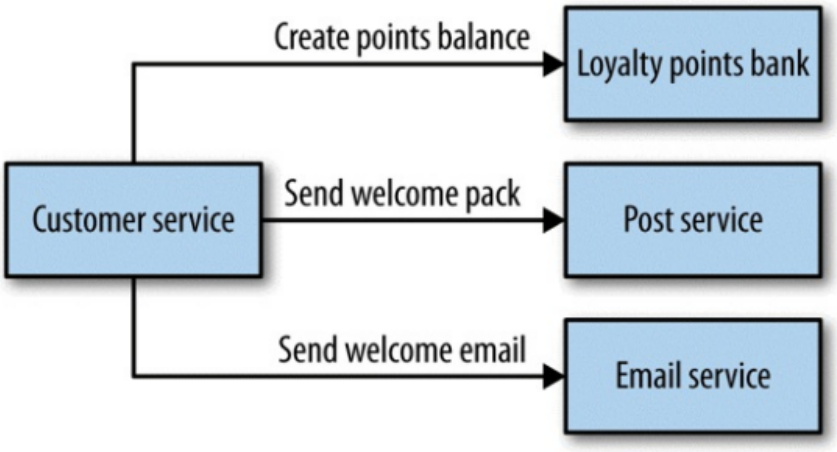

Figure 4-3. Handling customer creation via orchestration

Figur 1: [1, p. 91]

For at forstå orchestration bruges oftest en analogi til den virkelige verden hvor man kigger på et orkester. I et orkester er der en person der orkestrerer orkestret, nemlig dirigenten. Dirigenten sørger for at alle i orkesteret rammer deres toner på de rigtige tidspunkt og aggregere alle instrumenters toner til at få leveret et stykke musik. Med denne analogi kan vi kigge på services som de enkelte instrumenter, og dirigenten som en Saga der sørger for at aggregere og præsentere data fra forskellige services. Et simpel eksempel kunne være en request/response hvor en klient beder om at få alle informationer om en kunde og kundens betalinger baseret på kundens id. Saga'en vil requeste kundeservicen der kan returnere detaljer om kunden og requeste betalingsservicen om alle betalinger oprettet af kunden. Saga'en vil så aggregere dette data og præsentere det i et response. En definition på en orkesteringssaga kunne være følgende.

Orchestration saga - I en orkesterings baseret saga centraliserer man koordinations logikken i en enkelt service. Denne service står for at sende beskeder afsted til services der fortæller dem hvilke funktioner de skal udføre og eventuelt aggregere data til et response[2].

3.3 Choreography saga

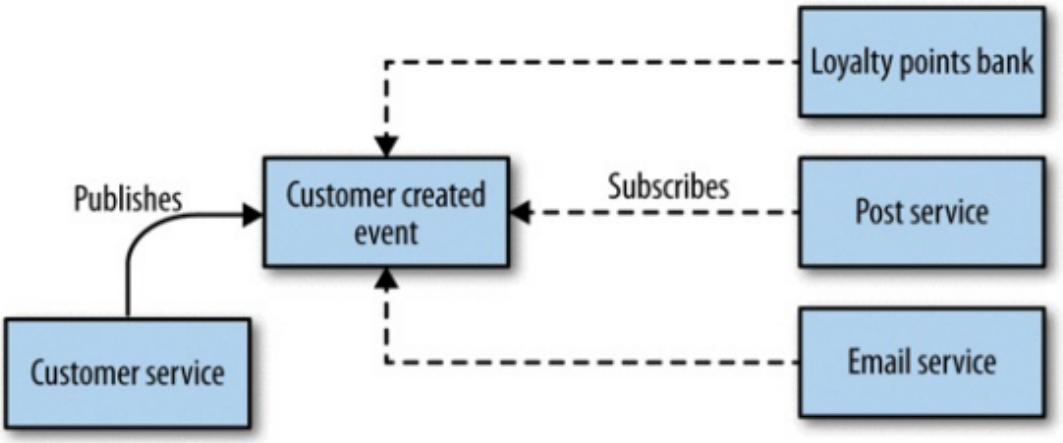

Figure 4-4. Handling customer creation via choreography

Figur 2: [1, p. 92]

4 2. delspørgsmål

4.1 Consumer-Driven Contracts

5 3. delspørgsmål

5.1 Hvad er eventual consistency?

5.2 Hvordan tager man højde for eventual consistency?

Litteratur

[1] Sam Newman. *Building microservices: designing fine-grained systems*. "O'Reilly Media, Inc.", 2015.

[2] Chris Richardson. *Microservices Patterns: With Examples in Java*. Manning Publications, 2019.