Microservices og Availability

Adam Lass og Rasmus Helsgaun

18. december 2020

Abstract

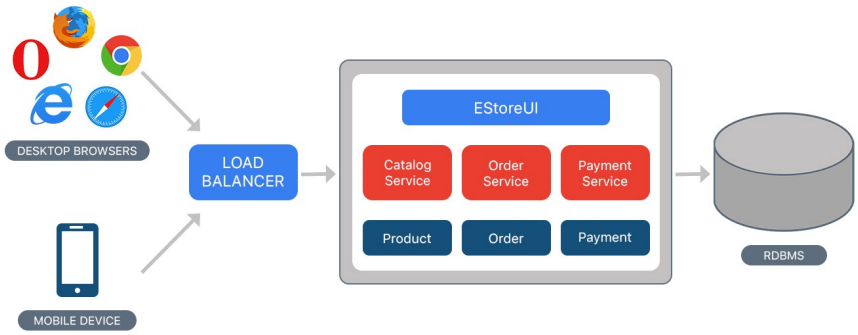
Ensuring high availability in a monolithic application is almost impossible due to its deployment and scalability challenges. This poses a big threat to businesses with a monolithic system today, that want to scale tomorrow. In this article we will explore how Microservices attempts to solve these challenges, and at what cost. By implementing this architecture when appropriate, businesses stand to win big on flexibility, scalability and availability of their systems.

Hvad indebærer availability?

Når der er tale om availability, refereres der til tilgængeligheden af software og dets evne til at udføre en opgave når der er brug for det. Konceptet bygger på reliability, som handler om hvor pålidelig et system er, og inkorporerer tilmed også konceptet om gendannelse af systemet ved fejl. I denne artikel vil der blive set nærmere på hvordan Microservices kan skabe værdi for en kunde i form af availability og hvilken bekostning det har sammenlignet med en monolitisk arkitektur.

Hvad er en monolitisk arkitektur?

Den monolitiske arkitektur beskriver en softwareapplikation, hvortil alle komponenter er samlet i et enkelt system.



Figur 1: *Illustrerer opsætningen af den monolitiske arkitektur, med alle systemets komponenter samlet i ét modul bagved en load balancer.[1]*

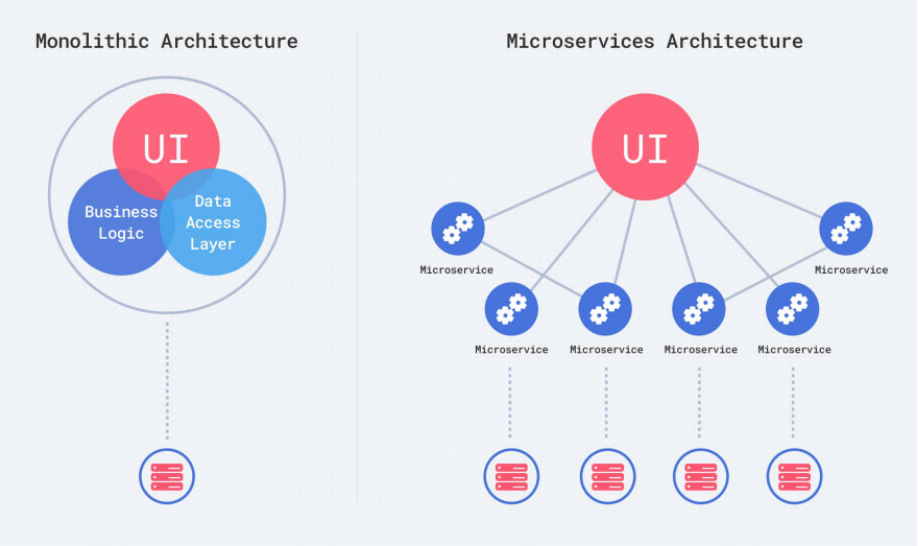
Fordelene ved at bruge en monolitisk arkitektur er blandt andet, at det i de tidlige stadier kan være enkelt at udvikle, deploye og teste, da man har alle komponenter samlet i én applikation.[2] Givet at ens monolitiske system er udviklet stateless, er det også forholdsvist simpelt at skalere horisontalt, ved at køre flere instanser af systemet bag en load balancer, som illustreret på **figur 1**.[1]

Der er dog også en række ulemper ved at udvikle og vedligeholde et monolitisk system, som bør overvejes inden den vælges som arkitektur. Eksempelvis kan det være sværere at vedligeholde et monolitisk system, da størrelsen af komponenterne, og afhængighederne derimellem, kan øge systemets kompleksitet. Den øgede kompleksitet medfører ofte en høj kobling, hvilket forværrer fleksibiliteten og overskueligheden af systemet.[3] Dette kan have store negative implikationer for udefrakommende udvikleres forståelse og overblik af systemet, hvilket kan resultere i øgede tidsmæssige og økonomiske udgifter ved videreudvikling.[4]

Samtidigt kan størrelsen have betydning for en længere opstartstid, hvilket er essentielt at holde til et minimum under opdateringer eller efter at systemet er crashet. I sådanne tilfælde bærer arkitekturen i særdeleshed præg af at en lille opdatering eller crash af et komponent kræver et fuldt genstart af alle komponenterne.[5] Hvis ikke disse scenarier bliver håndteret ordentligt, kan det resultere i at arkitekturen har en stor negativ indflydelse på reliability og availability af applikationen. En måde at håndtere et crash på kan være ved at bruge horisontal skalering. Her kan services der stadig kører overtage trafikken for services der er i fejltilstand eller under genopretning. Tilmed kan et værktøj oven på dette, såsom Kubernetes' Rolling Updates[6], hjælpe med at bevare et systems availability under opdateringer.

Hvad er Microservices?

Microservices er en arkitektonisk stil der samler services der specialiserer sig i forskellige egenskaber, som tilsammen udgør en større applikation. Dette er et direkte modstykke til den monolitiske arkitektur, hvor alle egenskaber ligger i samme service.



Figur 2: *Illustrerer forskellen mellem den Monolitiske arkitektur og Microservices arkitekturen.[7]*

Den opdelte arkitektur set i **figur 2** bringer en række fordele til områder af udviklingen i form af availability, scalability, testability, deployment m.fl., men har samtidigt også en række ulemper med sig.[1] En meget dominerende fordel ift. availability er eksempelvis hvordan Microservices gerne skal være små og afgrænset i forhold til deres egenskaber.[8] Dette har nemlig den betydning at potentiel downtime på den enkelte service bliver mindre alvorlig, da den nu kun eksisterer som en mindre del af et større system. Sådan en downtime kan være forårsaget af bugs der får servicen til at crashe, men også af almindelige opdateringer af servicen, begge ting som er forventelige i dagligdagen. Størrelsen af en Microservices bringer også den fordel, at man kan skalere de enkelte services horisontalt ud fra behov. Dette kan have en positiv indflydelse på ens hosting omkostninger og opstartstid, da man kan undgå unødvendig skalering af ressourcer.[9]

Det er dog også værd at nævne, at man ved brug af den alternative Nanoservices arkitektur[10], via frameworks såsom serverless[11], yderligere kan optimere skaleringen ved at gøre den dynamisk ud fra det aktuelle behov

af de enkelte funktioner. Skal ens applikation kunne håndtere meget varierende trakfik, kan man derfor med brug af Nanoservices opnå en endnu mere agil skaleringsmodel, der både øger availability og reducerer de gennemsnitlige hosting omkostninger.[12] Her er det dog vigtigt at nævne det overhead, der ligger i at funktionerne, kun bliver brugt én gang, hvilket også har en negativ effekt på responstiden. Dette betyder også, at der samtidigt skal være nok variation i trafikken, før at man kan opnå de økonomiske besparelser, når man benytter sig af en Nanoservices arkitektur.

En betydelig fordel som Microservices bringer er dets testability. Selvstændigheden og isolationen af komponenterne gør det nemmere at skrive tests, da afhængighederne imellem komponenterne er mindre sammenlignet med et monolitisk system.[13] Givet det faktum at tests skaber bedre kode, der sjældnere fejler[14], kan man udlede, at bedre testability i overført betydning også skaber bedre availability.

I det store hele kan brugen af Microservices være fordelagtig i forhold til availability, men der ligger selvfølgelig også en række udfordringer forbundet med at implementere arkitekturen. Udviklingsmæssigt skal man som udvikler ofte bruge meget tid på at hver enkelt komponent kan stå alene. Dette indebærer ofte ting som authentication, authorization, logging, osv., alle ting som man i et monolitisk system kun skal implementere én gang.[15] En anden udfordring er at de forskellige komponenter ofte skal snakke med hinanden via protokoller såsom HTTP, hvilket tilføjer ekstra netværkstrafik og processering i eventuelt implementerede loadbalancers. Dette kan både resultere i længere svartider og større regninger hos cloud leverandøren, begge ting man gerne vil forsøge at undgå, især som en mindre virksomhed.[16]

Overgangen til Microservices

Trods de overstående udfordringer viser en undersøgelse udført af O'Reilly[17] i 2020, med 1502 deltagere[18], at 61% allerede har brugt Microservices i et år eller mere. I en anden undersøgelse lavet af Bernd Ruecker i 2018, med 354 deltagende virksomheder[19], ses det at der trods den ekstra udviklingbyrde ved Microservices, alligevel er en interesse i at bruge arkitekturen til at opnå hurtigere udviklingstid hos 60% af deltagerne. Dette er nok i høj grad præget af den ekstra fleksibilitet Microservices giver når der skal integreres med eksisterende systemer i de større virksomheder. Tilmed beskriver en artikel[3], udgivet hos divante[20], hvordan store tech giganter såsom Amazon, Netflix og Spotify startede ud med en monolitisk arkitektur, og siden hen har gået over til at bruge Microservices.

Konklusion

Det ses at den generelle udvikling i valg af arkitektur både i større og mindre virksomheder går i retningen mod at bruge Microservices. Dette skyldes i høj grad, at arkitekturen er bedre egnet til, at håndtere de udfordringer, der kommer med at bygge store systemer, som skal kunne håndtere meget brugertrafik. Valget kommer dog på en bekostning af ekstra udviklings- og responstid, men disse udgifter kan nemt opvejes af de fremtidige fordele, der ligger i at bruge Microservices såsom availability, scalability og maintainability. Den monolitiske arkitektur har dog stadig sine fordele for nystartede virksomheder, der udvikler mindre projekter, hvor der ikke er meget variation i trafikken. Her er den nemlig hurtigere ift. udviklings- og responstid, og ofte billigere at hoste. Når der skal vælges arkitektur, kommer det i sidste ende an på, hvilke behov man har her og nu. Disse behov kan over tid ændre sig, i takt med at produktet og dets interesse udvikler sig, dog vil det i de fleste tilfælde kun gå i retningen mod at bruge Microservices, og måske endda Nanoservices. Står man derfor, og skal vælge, og hælder mest til at bruge Micro- eller Nanoservices, kan man lige så godt gøre det her og nu, i stedet for i morgen.

Litteratur

[1] S. ul Haq. Introduction to monolithic architecture and microservices architecture. https://medium.com/koderlabs/introduction-to-monolithic-architecture-and-microservices-architecture-b211a5955c63.

[2] J. Kanjilal. Pros and cons of monolithic vs. microservices architecture. https://searchapparchitecture.techtarget.com/tip/Pros-and-cons-of-monolithic-vs-microservices-architecture.

[3] A. Kwiecień. 10 companies that implemented the microservice architecture and paved the way for others. https://divante.com/blog/10-companies-that-implemented-the-microservice-architecture-and-paved-the-way-for-othe

[4] P. Karwatka. Monolithic architecture vs microservices. https://divante.com/blog/monolithic-architecture-vs-microservices/.

[5] R. Mhetre. When to choose microservices architecture over monolithic? why? https://medium.com/@mhetreramesh/when-to-choose-microservices-architecture-over-monolithic-why-794aed04d8db.

[6] Kubernetes. Rolling update deployment. https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#rolling-update-deployment.

[7] A. Barashkov. Microservices vs. monolith architecture. https://dev.to/alex_barashkov/microservices-vs-monolith-architecture-4l1m.

[8] Ambassador. Microservices. https://www.getambassador.io/learn/kubernetes-glossary/microservices/.

[9] JOE NEMER. Advantages and disadvantages of microservices architecture. https://cloudacademy.com/blog/microservices-architecture-challenge-advantage-drawback/.

[10] Evan Klein. Nanoservices vs. microservices. https://logz.io/blog/nanoservices-vs-microservices/.

[11] Serverless. What is serverless? https://www.serverless.com/learn/manifesto/.

[12] W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly, and S. Pallickara. Serverless computing: An investigation of factors influencing microservice performance. https://www.cs.colostate.edu/~shrideep/papers/ServerlessComputing-IC2E-2018.pdf.

[13] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. M. F. Montesi, R. Mustafin, and L. Safina. Microservices: yesterday, today, and tomorrow. https://arxiv.org/pdf/1606.04036.pdf.

[14] O. Mikhalchuk. The importance of unit testing, or how bugs found in time will save you money. https://fortegrp.com/the-importance-of-unit-testing/.

[15] T.v. Vignesh. Building a boilerplate for microservices — part 1. https://medium.com/techahoy/building-a-boilerplate-for-microservices-part-1-166ce00f5ce9.

[16] M. S. Nyfløtt. Optimizing inter-service communication between microservices. https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2479192/18340_FULLTEXT.pdf?sequence=1.

[17] Oreilly. About us. https://www.oreilly.com/about/.

[18] Oreilly. Microservices adoption in 2020. https://www.oreilly.com/radar/microservices-adoption-in-2020/.

[19] Tom Smith. New research shows 63% of enterprises are adopting microservices architectures. https://dzone.com/articles/new-research-shows-63-percent-of-enterprises-are-a.

[20] Divante. About us. https://divante.com/about-us.