Monolithische versus Microservices softwarearchitectuur

Het kiezen van het juiste ontwerp voor je app-ontwikkeling

Willem L. Middelkoop
3 mrt. 2020

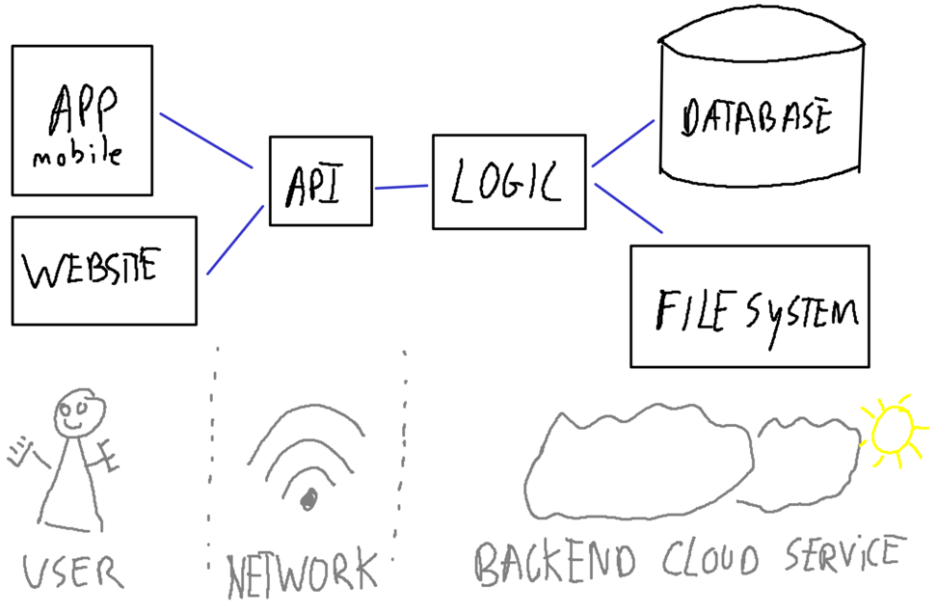
Deze week vloog ik naar Göteborg om mensen te ontmoeten van een grote internationale rederij, om te praten over de ontwikkeling van enterprise-level software. Tijdens de meeting waren er verschillende experts in de ruimte, een van hen vroeg me over het kiezen van de juiste software architectuur (voor grote, complexe, enterprise-level apps). Een zeer goede vraag, zeker een blogpost waard.

Bezoek aan een grote internationale rederij om te praten over bedrijfssoftware

Software-architectuur

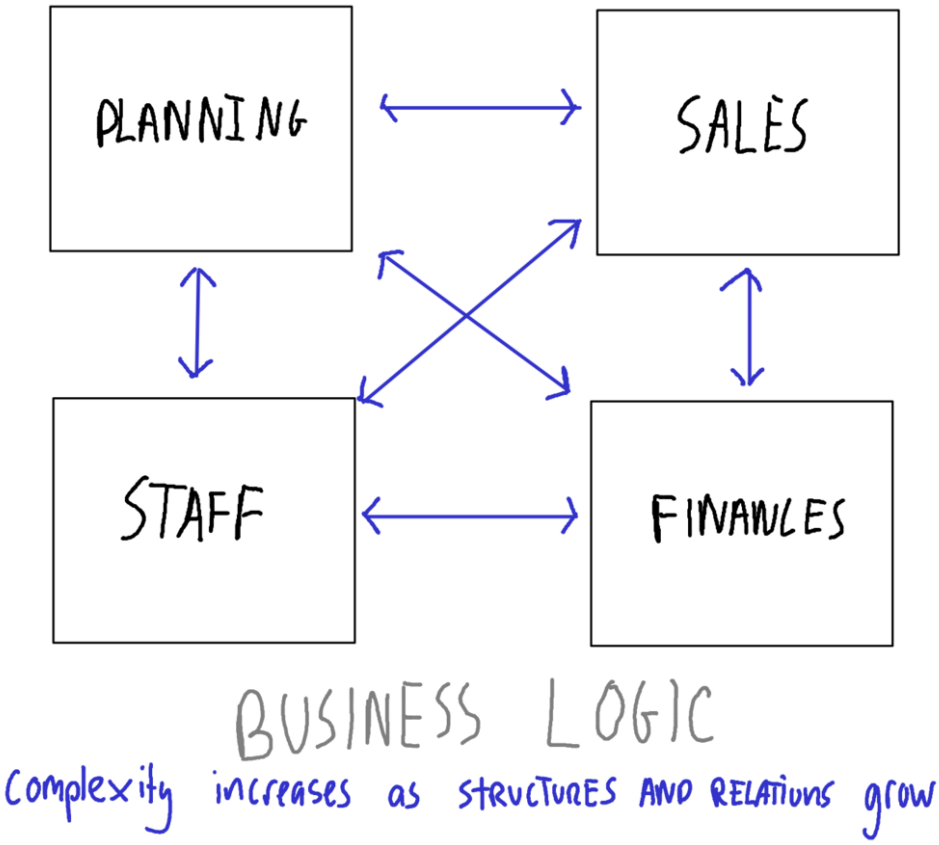
Wanneer je apps ontwerpt, moet je niet alleen nadenken over het uiterlijk, een goed ontwerp omvat ook hoe alles samenwerkt. Software-architectuur verwijst naar de fundamentele structuren van een softwaresysteem, elke structuur omvat elementen, relaties daartussen en eigenschappen van zowel elementen als hun relaties.

Het is analoog aan de architectuur van een gebouw. Zodra een architectuur is gedefinieerd en geïmplementeerd, wordt het kostbaar om deze achteraf te wijzigen. Het is daarom een goed idee om de juiste software-architectuur te kiezen *voordat* je begint met bouwen.



Eenvoudig voorbeeld van softwarearchitectuur - verschillende structuren die een systeem vormen met verschillende relaties tussen elkaar

Voor eenvoudige apps kan de software-architectuur voor de hand liggen, maar naarmate je applicatie groeit in omvang en functionaliteit, kan het aantal structuren in een systeem snel toenemen. Dit kan in de loop van de tijd gebeuren, naarmate de behoeften van je klant groeien. Hoe meer structuren, hoe meer relaties daartussen, hoe complexer het wordt.



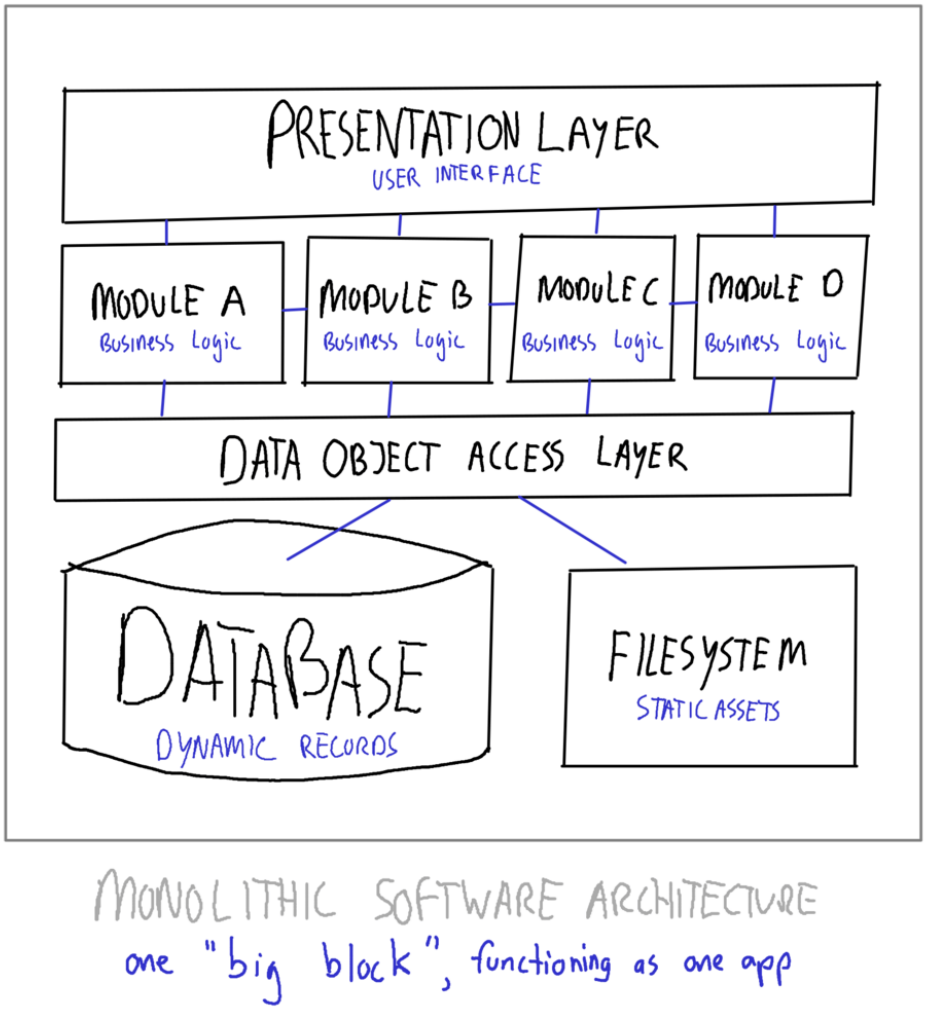
Verhoogde complexiteit naarmate het aantal structuren (en hun relaties) toeneemt

Omgaan met complexe softwareontwerpen

Als je verwacht dat je software veel verschillende dingen aankan, is de kans groot dat je te maken krijgt met complexiteit. Vanuit architectonisch oogpunt zijn er verschillende benaderingen om met deze softwarecomplexiteit om te gaan.

Monolithische Architectuur

Het beste omschreven als 'één groot blok' is de monolithische architectuur. Hier is alles wat je app doet, onderdeel van dezelfde codebase.



Monolithische softwarearchitectuur

In een ”monolithische architectuur” is de software vaak gelaagd, waarbij elke laag bestaat uit verschillende soorten componenten:

- **presentatielaag**: verantwoordelijk voor de gebruikers- en/of applicatie programmeerinterface (UI / API).
- **bedrijfslogica**: hier worden de daadwerkelijke bedrijfsregels gedefinieerd. Het is de kern van de applicatie die zeer nauw aansluit bij de daadwerkelijke bedrijfsprocessen.
- **data-objecttoegangslaag**: biedt een gemeenschappelijke interface tussen de actieve applicatie en persistente back-endopslag.
- **database en bestandssysteem**: hier worden alle gegevens en statische activa (bestanden, afbeeldingen, enz.) opgeslagen.

Hoe groter de applicatie wordt, hoe groter elke laag wordt. Dit maakt het moeilijker om een monolithische architectuur te schalen, op een gegeven moment worden dingen gewoon te groot en complex.

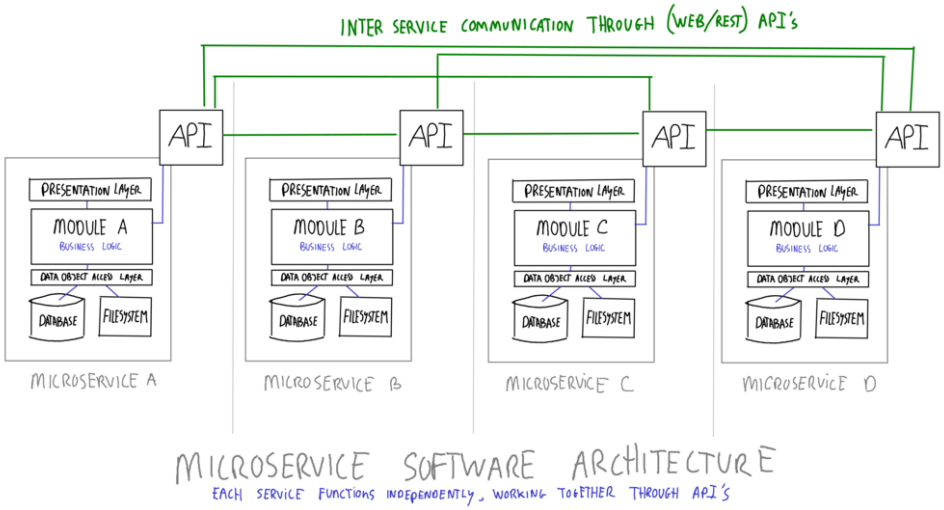
Elke keer dat je een wijziging aanbrengt in de applicatie, wordt de volledige codebase beïnvloed. Voor (zeer) grote applicaties maakt dit het moeilijker om de software te onderhouden, omdat het een volledig begrip van de hele applicatie vereist.

Omdat de hele applicatie onderling afhankelijk is, hebben problemen in één deel van de applicatie vaak aanzienlijke gevolgen voor het hele systeem. Communicatie tussen verschillende delen van het systeem vindt intern plaats, op code-niveau. Dit betekent dat de verschillende onderdelen in dezelfde (of compatibele) programmeertaal moeten worden geprogrammeerd. Dit is een belemmering om nieuwe technologieën (in de toekomst) te implementeren, omdat je de programmeertaal van een deel van het systeem niet gemakkelijk kunt wijzigen.

Microservices-architectuur

In plaats van één grote monolithische applicatie te bouwen, kun je je applicatie opsplitsen in kleinere onderling verbonden services. Elke microservice is een kleine applicatie op zich, compleet met zijn eigen (maar kleinere + eenvoudigere) architectuur, bedrijfslogica en datalagen.

De verschillende services communiceren met elkaar via API's die onafhankelijk van de gebruikte technologie zijn gedefinieerd, vaak in een gemeenschappelijke "API-structuur" zoals REST, JSON, Redis en XML. Hierdoor kan de combinatie van microservices functioneren als "één systeem".



Microservices Software Architectuur

Door de 'big block'-applicatie te ontleden in kleinere services, wordt het gemakkelijker om verschillende delen van het systeem te begrijpen. Dit maakt het gemakkelijker om het systeem te onderhouden, omdat elke microservice onafhankelijk kan worden ontwikkeld. Het verlaagt ook de drempel om nieuwe technologieën te implementeren, omdat je vrij bent om te kiezen welke technologie het meest geschikt is voor een bepaalde service. Eén deel van het systeem kan in een andere taal worden geprogrammeerd dan andere delen. Dit maakt het mogelijk voor verschillende teams (met verschillende expertise) om samen te werken.

De kunst en vaardigheid om de microservices-architectuur goed te krijgen, komt voort uit het vermogen om een microservice correct te definiëren. Te breed of te gedetailleerd kan de architectuur vernietigen. Je kunt bedrijfsprocessen, hiërarchische of domeinscheiding gebruiken om elke microservice te definiëren.

De uitdaging met de microservice-architectuur is dat het complexiteit aan het hele systeem toevoegt doordat het een gedistribueerd communicatiemechanisme vereist. De

verschillende API's die met elkaar verbinden, kunnen lastig zijn om te testen en te debuggen. Het wordt moeilijker om wijzigingen te implementeren die meerdere services omvatten.

Het definiëren, ontwerpen en implementeren van de verschillende interservice-API's vereist een overeenkomst over de uitgewisselde gegevens, dit kan moeilijk te bereiken zijn wanneer je van plan bent uitgebreide en niet-universele datastructuren te delen.

Het bewaken en implementeren van een microservice-architectuur kan complex zijn wanneer het aantal services toeneemt. Er is gewoon meer te beheren, wat meer planning en coördinatie vereist voor elk van de services.

De juiste architectuur kiezen

Mijn overleden vader vertelde me ooit dat je "het meest kunt verdienen aan de dingen die je niet doet". Het bouwen van complexe applicaties is inherent moeilijk, het beste wat je kunt doen, is proberen dingen eenvoudiger te maken door simpelweg niet alles te doen. Focus op de dingen die er echt toe doen, doe die dingen goed.

Als je applicatie eenvoudig en lichtgewicht is, is een monolithische architectuur de juiste keuze. Ontwikkeling, implementatie, onderhoud, alles is gemakkelijker als dingen klein zijn.

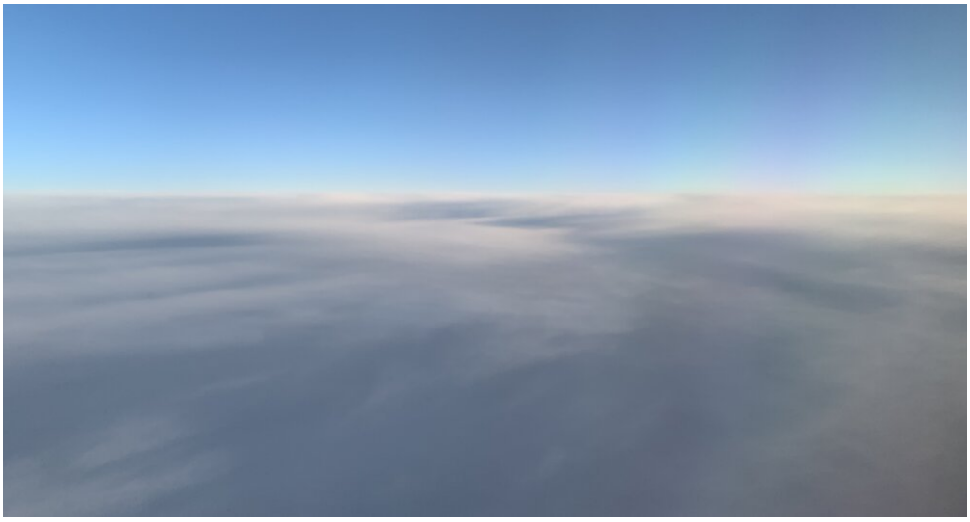
Als je complexiteit niet kunt vermijden, moet je zeker de microservice-architectuur overwegen. Stel jezelf de volgende vragen:

- Kun je een "gemeenschappelijke taal" definiëren tussen je microservices?
- Zijn er duidelijk verschillende bedrijfsprocessen?
- Hoe significant zijn deze bedrijfsprocessen onderling afhankelijk?
- Hoe verwacht je dat je applicatie zich in de loop van de tijd zal ontwikkelen, verwacht je dat deze zal groeien of krimpen?
- Hoeveel verschillende programmeertechnologieën ben je van plan te gebruiken? Verwacht je dat dit in de toekomst zal veranderen?
- Wie gaat de software bouwen en onderhouden, hebben ze ervaring met het beheren van een dergelijke architectuur?

Conclusie

Er is geen "wondermiddel" antwoord op de vraag welke software-architectuur juist is. Beide benaderingen hebben hun voor- en nadelen.

Het beste wat je kunt doen, is een bewuste - weloverwogen - beslissing nemen en je daarop vastbijten!



"The sky is the limit" als je de gekozen architectuur beheerst (verander deze alleen niet tijdens de vlucht!)