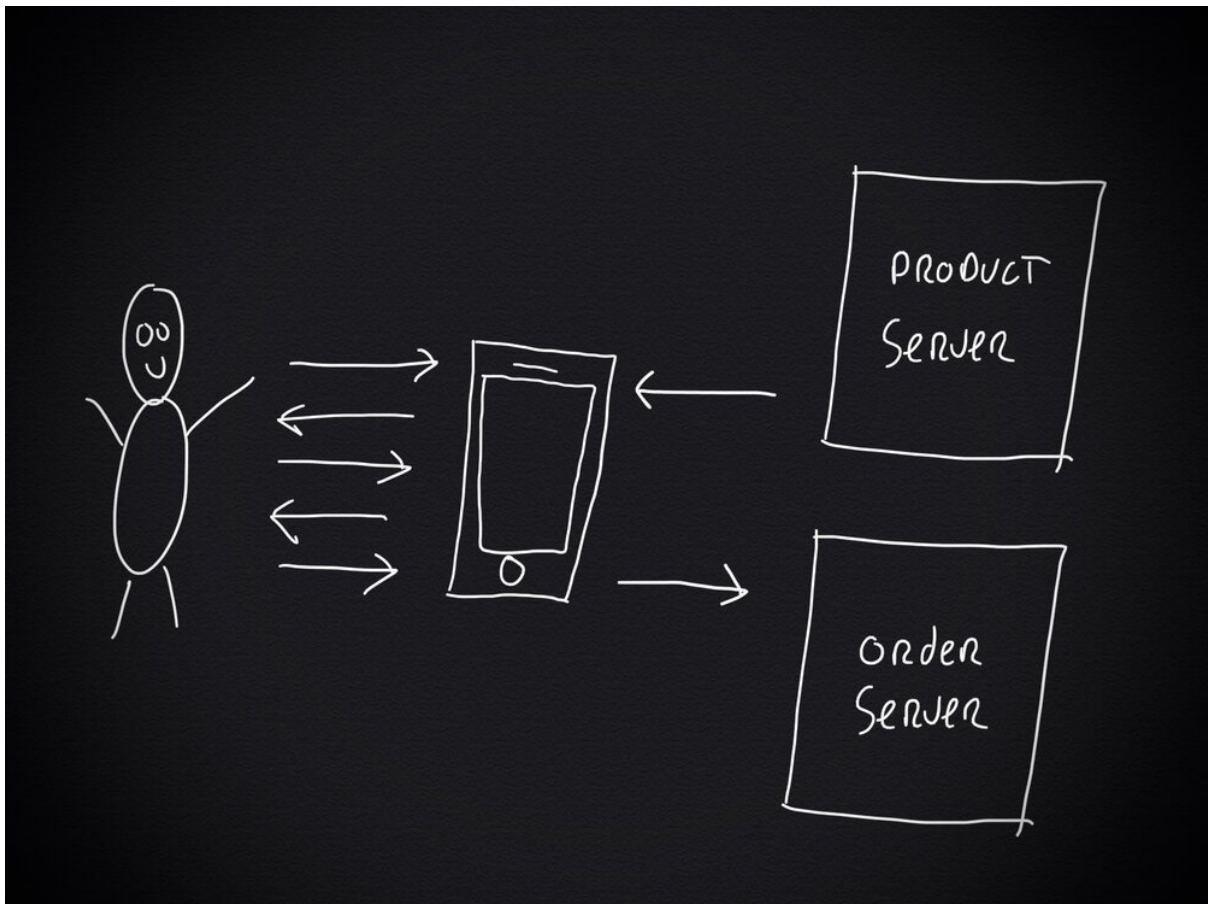


Schaalbaar applicatieontwerp zonder magie

Benutten van client rekenkracht voor hoge prestaties met veel gebruikers

Willem L. Middelkoop

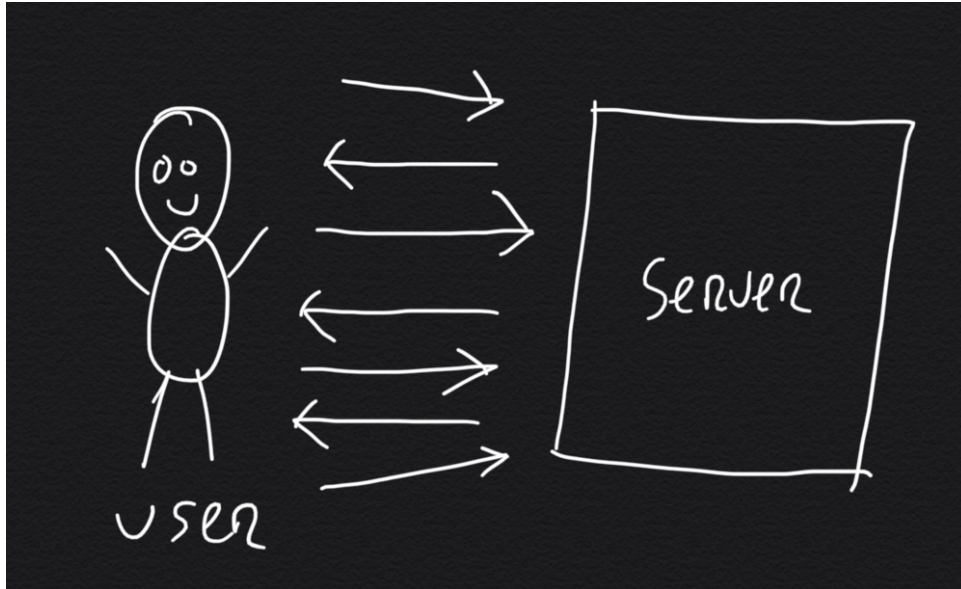
11 mei 2020



Als onderdeel van de online voedselbestel-app die ik aan het bouwen ben, moest ik een schaalbare backend-infrastructuur ontwerpen die veel gelijktijdige gebruikers aankan. Schaalbaarheid wordt beschouwd als een moeilijk probleem om aan te pakken. Vaak wordt het gepresenteerd als iets magisch, gedaan door bedrijven van miljoenen dollars met behulp van geheime tools. Maar, er bestaat niet zoiets als magie, of toch wel?

Wat is schaalbaarheid?

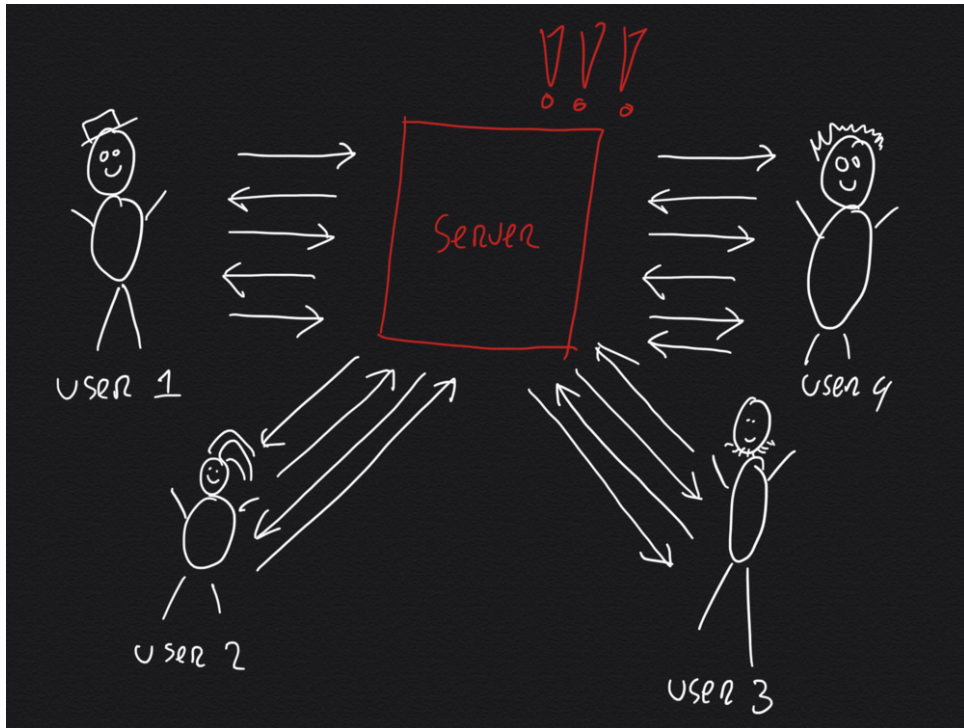
Als je een app bouwt, begin je waarschijnlijk heel klein. Slechts een paar gebruikers (of alleen jij, de ontwikkelaar) die met de app werken. Alles zal prima werken: de app werkt goed, snel en er zijn geen problemen. (*koester dit gevoel!*)



Eén gebruiker die interactie heeft met één server

In dit vroege en kleine scenario kan de enkele server waarop de app draait, de interactie van de gebruiker aan. Elke keer dat de gebruiker op een knop tikt of klikt, doet de enkele server wat werk. Als je goed programmeert, kunnen deze workloads worden geoptimaliseerd tot efficiënte stukjes werk. Dit doen is een typisch voorbeeld van een [monolithische software architectuur](#).

Zelfs als je project of app vrij klein is, kan er op een gegeven moment een toestroom van gebruikers zijn. Als je systeem geen hoge belasting aankan, is de kans op falen groot.



Server onder hoge belasting die meerdere gelijktijdige gebruikers afhandelt

Vanwege het [monolithische app ontwerp](#) werken alle gebruikers met dezelfde server. Al hun interactie (tikken, klikken, invoer) wordt door deze server afgehandeld. Met een verhoogde belasting moet het veel harder werken. Op een bepaald moment zul je merken dat dingen vertragen omdat de server de hoeveelheid werk die hij moet doen niet kan bijbenen. Dit betekent dat je app niet goed schaalbaar is, met andere woorden: je zit in de problemen.

Falen op 'het moment suprême'

Niet goed schalen is een groot probleem dat niemand mag onderschatten. De meeste apps (en hun bedrijfsmodellen) zijn afhankelijk van hoge volumes met weinig inkomsten per individuele gebruiker. Als je wilt dat je app een (financieel) succes wordt, moet je ervoor zorgen dat deze goed werkt wanneer er een plotselinge toename van verkeer is. Als je app crasht wanneer deze plotseling de aandacht van de massa krijgt, mis je een 'once-in-a-lifetime'-kans voor (organische) groei.

In zekere zin is het ontwerpen voor schaalbaarheid als het vastmaken van je veiligheidsgordels wanneer je in je zelfgebouwde raket zit, met experimentele motoren. Je weet nooit precies wanneer hij zal ontsteken, maar als het gebeurt, kun je er maar beter voor zorgen dat je stevig vastzit (en geniet van de rit!).

Veelvoorkomende manieren om schaalbaarheid te bereiken

Er is veel geschreven over het bouwen van schaalbare apps, veelvoorkomende manieren om de schaalbaarheid te verbeteren zijn:

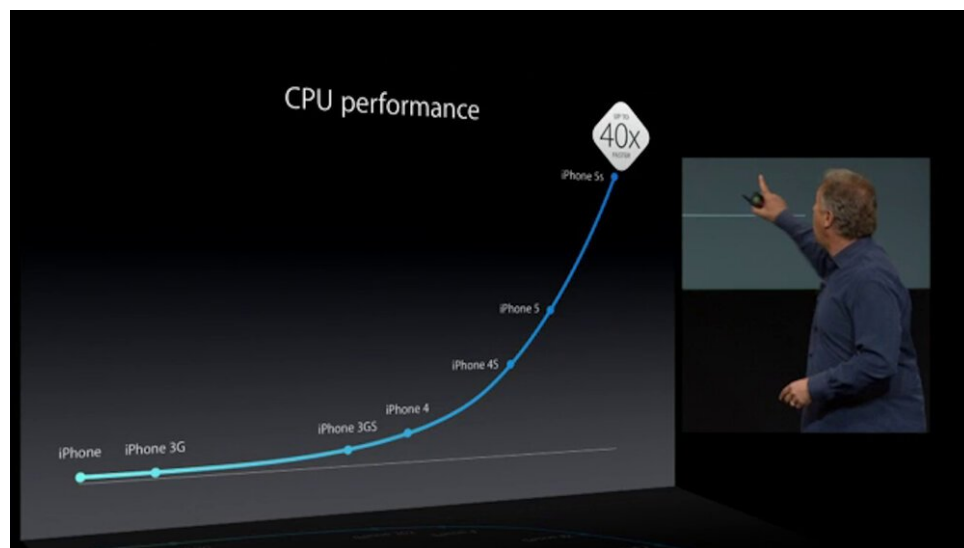
- **Servercapaciteit verhogen:** meer geheugen, CPU-kracht, opslag toevoegen. Dit zal werken, maar het zal je maar 'zo ver' brengen.

- **Meer servers toevoegen:** in plaats van een enkele server die als bottleneck fungeert, kun je meer servers toevoegen die de workload zullen delen. Dit is gemakkelijker gezegd dan gedaan, omdat het load balancing en data sharing vereist, wat lastig kan zijn (bijv. als je een orderteller hebt en twee servers tellen, welke bepaalt dan het volgende nummer?)
- **Programmeercode optimaliseren:** gebruik de juiste tools! Kies een prestatiegerichte programmeertaal en dito serversoftware. Kijk naar functioneel programmeren om je code asynchroon op meerdere cores te laten draaien. Maak het stateless. Benchmark het. Optimaliseer het. Elke milliseconde die je wint op een enkele request telt snel op wanneer je grote volumes verwerkt.
- **Gebruik de database en scheid deze van de applicatieserver:** Als je een database gebruikt, maak er dan gebruik van! Profiteer van query caching, indexen en zoekmogelijkheden. De meeste database-oplossingen bieden beproefde clusteringopties; probeer niet het wiel opnieuw uit te vinden dat anderen al hebben geperfectioneerd.

Het beste wat je kunt doen, is al deze opties overwegen. Zelfs als je niet meteen meer servers toevoegt, moet je je code zo schrijven dat deze later wordt ondersteund. Het benchmarken en optimaliseren van je code moet een integraal onderdeel van je werk zijn, niet alleen een achterafgedachte.

Gebruik maken van client-rekenkracht

Wanneer je alle gebruikelijke manieren hebt overwogen om schaalbaarheid te bereiken, is er "nog één ding".

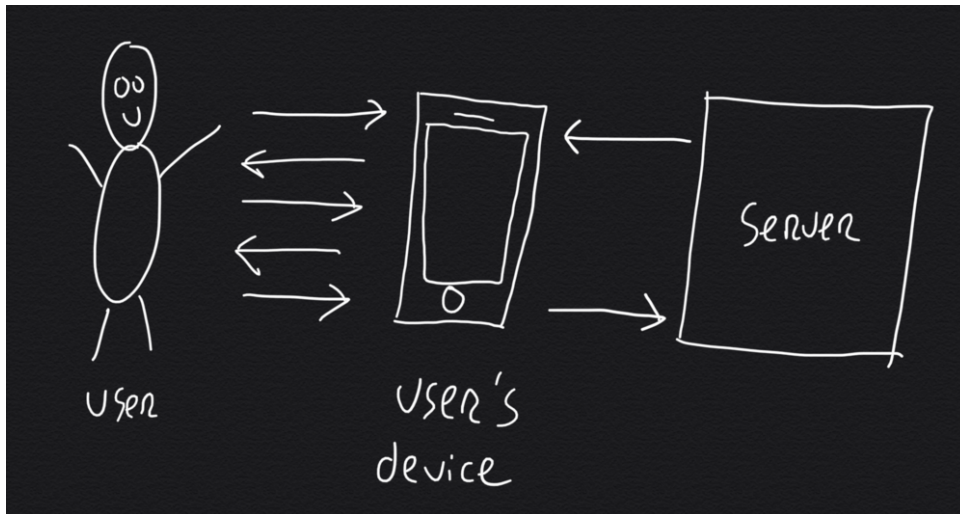


Smartphones (en tablets, pc's) zijn in de loop der jaren krachtiger geworden (image: Apple)

Weinig ontwikkelaars overwegen dit: het schaalbaarheidsprobleem bevat de oplossing! Wanneer je 1000 gebruikers hebt, heb je 1000 computers met krachtige CPU's en veel geheugen. Moderne computers en smartphones zijn steeds capabeler geworden.

Je hoeft alleen maar een manier te vinden om de rekenkracht van de client te benutten. Hoewel het misschien een kwaadaardig iets klinkt (het gebruiken van de rekenkracht

van de gebruiker), zal het hen een veel betere, snellere ervaring bieden. Het beste is dat *hun* rekenkracht zal worden besteed aan *hun* fantastische ervaring, waardoor je schaalbaarheidsprobleem als een prettig neveneffect wordt opgelost.



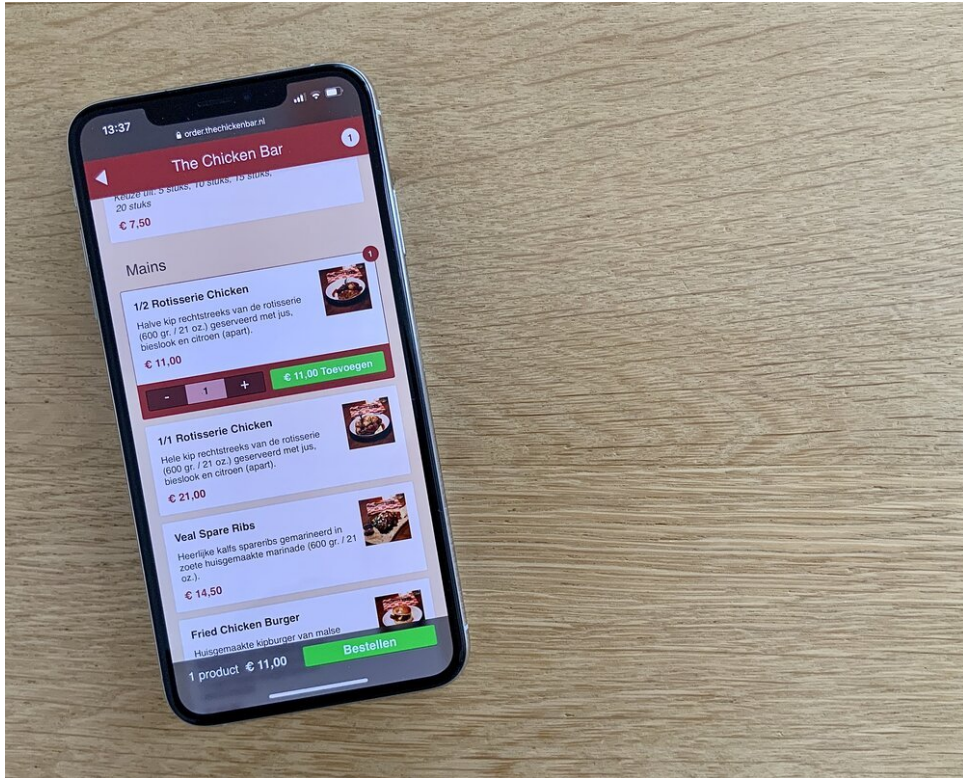
Gebruikmaken van het apparaat van een gebruiker: de telefoon van de gebruiker doet het meeste werk dat oorspronkelijk door de server werd gedaan

Door gebruik te maken van de rekenkracht van de gebruiker, verminder je de hoeveelheid werk die de server moet doen. In plaats van dat de server *elke* keer dat de gebruiker op een knop tikt of klikt een beetje werk doet, wordt dit werk nu grotendeels gedaan door het apparaat van de gebruiker. Dit is efficiënter dan communiceren via een wifi/4G-verbinding met een server (bespaart batterijvermogen en tijd).

Het vereist een beetje heroverweging, maar dit principe kan ook worden toegepast op webapplicaties. In plaats van alles op de server af te handelen, kun je veel werk op de client uitvoeren. Dingen zoals zoeken, browsen, filteren, navigeren en zelfs dingen toevoegen aan een winkelwagentje kan worden gedaan met behulp van client-side JavaScript. Jij - beste ontwikkelaar - moet zoeken naar manieren om dit te laten werken, maar als het je lukt, heb je bijna oneindige schaalbaarheid binnen handbereik.

In de praktijk: app voor het bestellen van eten

Voor de [app voor het bestellen van eten](#) heb ik gezocht naar manieren om de rekenkracht van de client te benutten om hoge prestaties te bereiken. De grootste uitdaging bij bezorgen en afhaalbestellingen is dat ze pieken rond etenstijd. Veel mensen gebruiken de app rond dezelfde tijd, dit is een recept voor schaalbaarheidsproblemen.

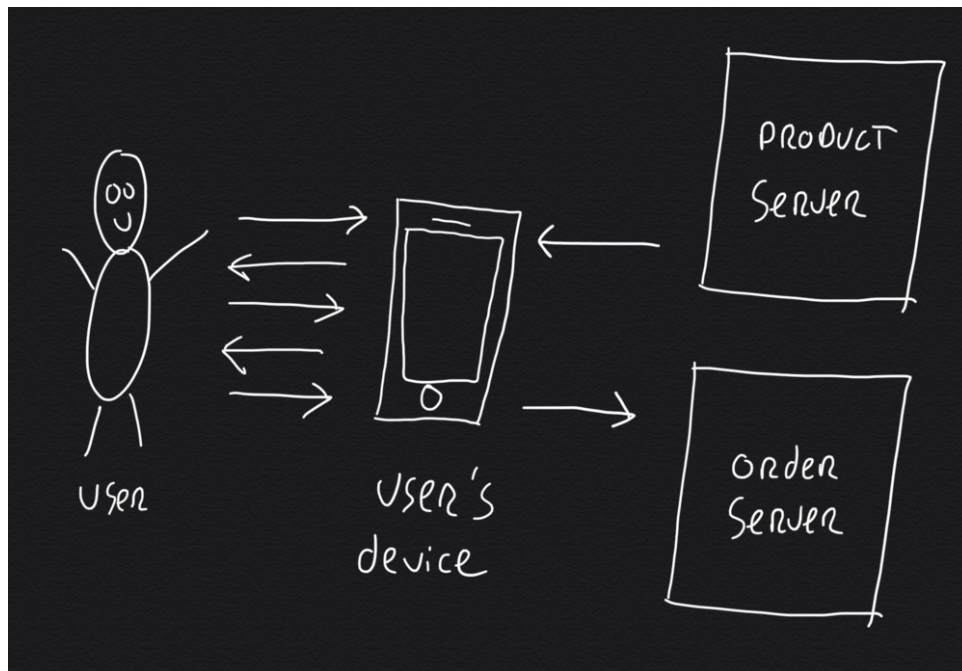


Online eten bestellen

Als je denkt aan het online bestellen van eten, kun je het hele proces opsplitsen in kleinere stukjes:

- **1) De pagina openen:** de app/pagina laden
- **2) Bladeren door de verschillende opties:** de producten weergeven, hun beschrijvingen, prijzen, foto's, enz. bekijken.
- **3) Zoeken naar iets specifiek:** zoeken op categorie, productnaam, enz.
- **4) Een product selecteren:** het toevoegen aan je bestelling
- **5) Een product aanpassen:** een saus, bijgerecht, topping, enz. selecteren.
- **6) Je gegevens invoeren:** je naam, telefoonnummer, bezorggegevens, enz.
- **7) Betalen voor je bestelling:** een betaalmethode selecteren, verbinding maken met je bank of creditcard
- **8) De app sluiten**

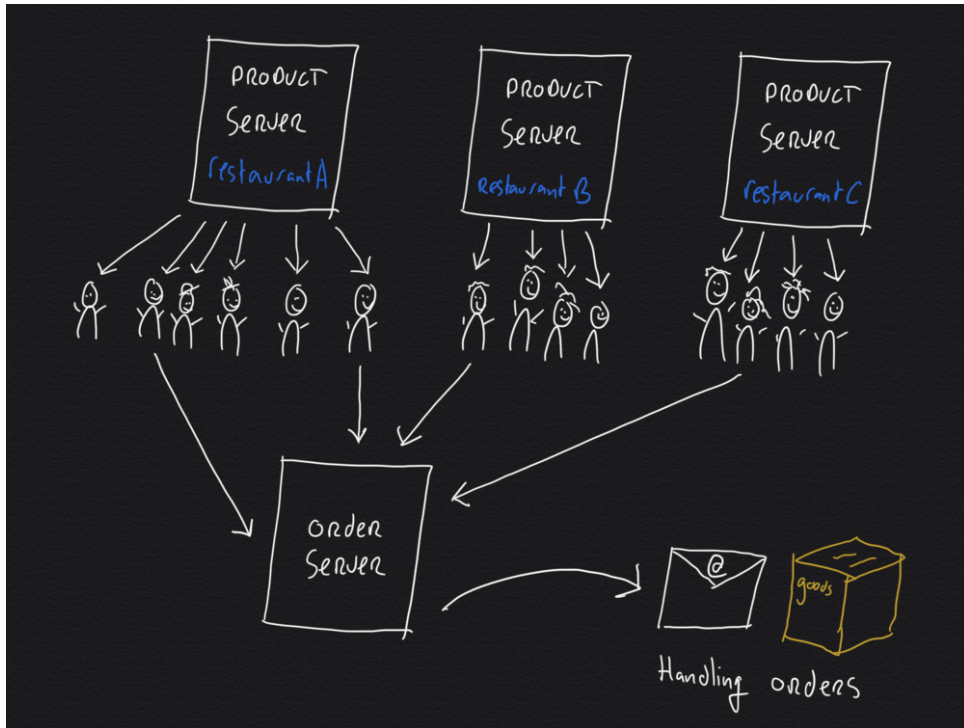
De meeste van deze stappen kunnen zo worden geprogrammeerd dat de server *niet nodig* is. Alleen stap 1 (**laden**) en stap 7 (**betalen**) vereisen contact met de backend-infrastructuur. Het is belangrijk om te beseffen dat stap 2 tot en met 5 meerdere keren worden herhaald, omdat het gebruikelijk is dat gebruikers meerdere producten aan hun bestelling toevoegen.



De resterende serverbelasting scheiden: producten aanbieden en bestellingen verwerken

Door de resterende workload per taak te scheiden, kan de software verder worden geoptimaliseerd. De "**productserver**" is geoptimaliseerd om statische activa (teksten, prijzen, afbeeldingen) te serveren waarmee de client de productbrowse-ervaring kan creëren. Je kunt dit soort server sterk optimaliseren door gebruik te maken van caching, HTTP/2, compressie, enz.

Alleen die gebruikers die het bestelproces voltooien, worden doorgestuurd naar de "**bestels**erver" die de betaling afhandelt. Deze maakt verbinding met de betalingsprovider. Zodra een betaling is voltooid, neemt de betalingsprovider contact op met de bestels^erver om deze te informeren over de betalingsstatus. Dit is iets waar je volledige (server-side) controle over wilt hebben, zoals ik deed toen ik eerder [een betalingssysteem ontwierp](#). De resterende workload op de "bestels^erver" is veel kleiner omdat niet iedereen een bestelling plaatst, en veel van het orderverwerkingswerk kan op de achtergrond worden gedaan.



Veel verkeer afhandelen via gedistribueerd computergebruik

Om de schaalbaarheid te maximaliseren, kun je eenvoudig meerdere servers toevoegen (of een content delivery network gebruiken) om de statische activa te serveren. Voor de app voor het bestellen van eten gebruik ik een aparte "productserver" *per restaurant*. Dit zorgt voor hoge prestaties en verkort de laadtijden. Mensen merken dat het snel is, sommigen kunnen niet geloven dat het webtechnologie is, het is *bijna* magisch.

Conclusie

Het ontwerpen van een schaalbare applicatie kan worden gedaan zonder magie of miljoenenbudgetten. Je moet er gewoon goed over nadenken en zoeken naar kansen die zich voordoen, samen met de uitdagingen.

Het probleem bevat de oplossing. Omgaan met veel gelijktijdige gebruikers is niet alleen een probleem: ze brengen hun eigen rekenkracht mee, het is de oplossing! Je hoeft het alleen maar te gebruiken.