Connecthing Architectuur Review

Ernst Vorsteveld, Stern it-consultants

19-12-2021

# Inleiding

Connecthing staat voor een aantal uitdagingen. Door eigen ontwikkeling en overnames, is een veelzijdig applicatielandschap ontstaan, zowel technologisch (PhP, .NET) als ook functioneel (SalesViewPro, Nutselect, VanZelf).

Mijn opdracht is het reviewen van de technische architectuur van het platform dat in ontwikkeling is en moet leiden tot een stroomlijning van de gebruikte technologie. Dit platform moet zowel het partner en leverancier kanalen (Nutselect) als het consumer kanaal (Vanzelf) ondersteunen. Drie vragen moeten worden beantwoord:

* Kan met de architectuur de doelstelling nieuwe landen te on-boarding (zo eenvoudig mogelijk) worden gerealiseerd?
* Is de architectuur afgestemd op de te realiseren functionele scope en eisen aan productie?
* Leidt de architectuur tot een goed te onderhouden en uit te breiden system?

Dit document geeft een (kort) overzicht van de bevindingen.

# Samenvatting

Donderdag 3 december 2020 heb ik Connecthing in Breda bezocht en gesproken met Roy Kremer en Mathieu Bruning. In de ochtend hebben we gesproken over de architectuur van de systemen in ontwikkeling, NutSelect en Vanzelf. In de middag hebben we gesproken over een strategie om V1.5 op te leveren eind Q2. Mathieu heeft twee presentaties getoond. Daarnaast heb ik op donderdag 10 december 2020 een meeting met het development team bijgewoond. Samengevat zijn mijn bevindingen hieronder weergegeven.

Vandaag zijn er grofweg 2 technology stacks in gebruik, te weten:

* Synnion platform: PhP
* Nieuw platform (SalesViewPro 2.0 & Connecting Cloud): .NET

Er gewerkt aan het nieuwe platform waarmee twee business units worden bediend, Nutselect en een nieuw product Vanzelf. Dit platform wordt gerealiseerd met een aantal development technolgieen:

* .NET
* Blazor front-end
* Vue.js front-end

Er is een diagram gemaakt van de doelarchitectuur, waarin een schematisch overzicht van de doelarchitectuur wordt weergegeven.

Op donderdag 3 december was de exacte scope van de oplevering per einde Q2 2021 niet bekend bij CTO en development team. Op donderdag 10 december was er meer duidelijkheid omtrent de scope: huidige functionaliteit (1.0) plus de pijnpunten (aanvraagstraat, productbeheer, content control, facturatie). De exacte omvang in detail is helaas evenmin duidelijk. De vraag is hoe een accurate inschatting kan zijn gemaakt voor genoemde opleverdatum. De scope zal zeker verder uitgewerkt moeten worden.

Ten aanzien van de 3 vragen uit de opdracht het volgende. Om de doelstellingen die Connecthing heeft met het te ontwikkelen platform bestaande uit Nutselect, Vanzelf en een set gedeelde componenten worden onderstaande acties aanbevolen.

Software ontwikkelteams zullen succesvol zijn als zij werken met een duidelijke scope en doelstelling. Om dit te bereiken is er verdere verdieping op kwaliteitsaspecten van een software ontwikkeling proces noodzakelijk. De scope (functionaliteit) zal glashelder moeten zijn voor alle betrokkenen. De scope moet worden samengevat in een roadmap en gedetailleerd in een backlog van epics en stories (scrum). Daarnaast is gebleken dat development teams behoefte hebben aan domeinkennis. Aan de andere kant zal de architectuur verder moeten worden uitgewerkt. Om te beginnen met het vastleggen van de architectuur uitgangspunten. Van daaruit kan de doelarchitectuur worden vastgesteld en vastgelegd. Daarna is detaillering noodzakelijk. Teams zouden niet mogen starten met ontwikkelen, voordat die aspecten helder zijn. Als toch wordt gestart, leidt dat tot systemen die te laat worden opgeleverd en waarin relatief veel technical debt zal worden aangetroffen. Als laatste wordt opgemerkt dat het nemen van een besluit over hoe de noodzakelijke functionaliteit voor Q2 kan worden opgeleverd, van groot belang is. De aangetroffen kwaliteit van de vastgelegde architectuur en dan met name de detaillering daarvan zal snel moeten worden verhoogd. Hulp van buiten zal zeker helpen bij het verhogen van het niveau van de architectuur. Het gebruik van formele technieken is hierbij zeer gewenst. Het draagt bij aan de mate waarin architectuur kan worden gecommuniceerd en gebruikt.

## Positief

De positieve bevindingen zijn:

* Op detail nivo zijn er in het ontwikkelteam afspraken gemaakt:
  + Code style standaarden;
  + Design pattern "Clean Architecture" adoptie;
* Er is gestart met de realisatie van CI/CD pipeline;
* Er wordt hard gewerkt aan de doelarchitectuur in archimate.

## Kan Beter

Verbetering is mogelijk ten aanzien van:

* Gebruikte technieken:
  + Er wordt een keuze gemaakt voor twee front-end technologieen;
  + Binnen het team bestaan vragen omtrent de techniek te gebruiken. De te gebruiken techniek moet worden vastgesteld;
  + Blazor stelt eisen aan de browser van de gebruiker;
  + En leidt tot meer complexiteit.
* Doel architectuur:
  + Timing: Het architectuur diagram bestaat sinds kort;
  + Het diagram is niet up-to-date;
  + Voldoende detaillering ontbreekt voor ontwikkelteams;
  + Er wordt geen formele tekentechniek gebruikt.
* Architectuur patterns ontbreken (bijvoorbeeld):
  + Event gebaseerd versus request gebaseerd;
  + Storage strategie is onduidelijk en er bestaat verschil van inzicht (data context);
  + Impact SQL versus NoSQL storage onduidelijk;
  + Strategie voor microservices versus miniservices moet verder worden uitgewerkt;
* Het deployment model:
  + Een deployment diagram ontbreekt;
  + CI/CD pipeline stopt bij staging;
  + De eisen ten aanzien van de beschikbaarheid lijken niet te worden gerealiseerd;
  + High availability: er is een shared stack die high availability moet ondersteunen.
* De scope:
  + Er is een oplevering gepland einde Q2, terwijl de scope onduidelijk is;
  + De epics en stories ontbreken, zijn niet compleet, of te onduidelijk voor ontwikkelaars;
  + Voldoende en grondige domeinkennis ontbreekt in de teams.
* Maatregelen voor de operatie:
  + Monitoring is mij niet duidelijk geworden;
  + Requirements en strategie ten aanzien van auto scaling en auto recovery zijn onduidelijk.

# Technieken

Er wordt gebruik gemaakt van VUE.js en Blazor voor ontwikkeling van user interfaces. Welke technologie beter of geschikter is voor het doel is minder belangrijk dan het feit dat er 2 worden gebruikt. De aanbeveling is om het aantal technolgieen die in gebruik zijn te minimaliseren, zeker gezien de omvang van het ontwikkelteam. Het kiezen van een technologie is belangrijk, omdat de discussie over de techniek wordt gestopt. Daarnaast zorgt duidelijkheid voor focus en geeft mensen de energie zich te verdiepen in de gekozen optie. Het is belangrijk dat de te gebruiken techniek wordt vastgesteld en vanaf dat moment gebruikt.

Ik merk op dat Blazor relatief nieuwe technologie is en pas sinds mei 2020 gereed is voor productie. Dat een C# ontwikkelaar er snel zijn weg ik kan vinden is duidelijk en kan een goede reden zijn deze techniek te omarmen. Aan de andere kant is het ontwikkelen van een user interface een vak apart.

# Doel architectuur

Architectuur begint bij het formuleren van uitgangspunten. De uitgangspunten worden gebruikt voor het ontwerpen van de architectuur, maar ook bij het nemen van ontwerpbeslissingen door developers. Een goed voorbeeld is "The Twelve-Factor App" (zie <https://12factor.net/>).

De indruk bestaat dat er pas sinds kort een diagram bestaat waarop min of meer alle componenten in samenhang worden getoond. Dat betekent niet dat er niet een goed plan is geweest, maar wel dat er geen architectuur is gevormd en vastgelegd. Het hebben van een overzicht nodigt uit tot discussie en verdieping. Met een klein team, waarin iedereen op de hoogte is van de scope en aanpak, is dit minder een probleem dan bij een groter team met nieuwe leden zoals nu het geval bij Connecthing. Gebrek aan architectuur en gedeelde visie leidt veelal tot minder optimaal software design en technical debt. Het verdient aanbeveling die architectuur te ontwikkelen en te verdiepen. Gebleken is dat het beschrijven van de architectuur in archimate diagrammen is opgepakt.

Verder valt op dat het diagram niet up-to-date is: op een aantal punten klopt het niet, wat tot verwarring zal leiden. Ook ontbreken componenten die een sleutelrol spelen bij de implementatie, zoals bijvoorbeeld een component voor messaging.

Door discussie wordt een architectuur gedeeld en aangepast met goede ideeen van iedereen. Verdieping en verduidelijking in de architectuur wordt gerealiseerd, omdat wordt herkend waar hiaten zitten of waar architectuur kan en moet helpen. Het zal helpen bij de adoptie en waardering van developers.

Een microservices architectuur, volgens de theorie uitgevoerd, is niet eenvoudig. Er zijn vele verhalen te vinden op het internet en evenzovele strategieen. Wat wel van belang is dat er overeenstemming over bestaat in het ontwikkelteam. Die lijkt te ontbreken. Of er nu micro-, macro- of miniservices worden gebruikt maakt niet uit, als er maar voor wordt gezorgd dat data altijd door een enkele service wordt beheerd en geleverd. Ook hierover bestaat geen overeenstemming heb ik de indruk. Hier dient de strategie verder uitgewerkt te worden.

Het diagram dat er nu is kan het beste omgezet worden in een diagram waarin gebruik gemaakt wordt van een formele techniek, bijvoorbeeld het veel gebruikte framework genaamd Archimate (zie <https://www.opengroup.org/archimate-forum/archimate-overview>). Daarnaast is het gebruik van UML Sequence diagrammen aan te bevelen.

Ik constateer verder dat we niet gesproken hebben over onderstaande aspecten. Wellicht door gebrek aan tijd:

* Detaillering: meer detail in de architectuur leidt tot beter software design en begrip;
* Deployment model: het hebben van een overzicht van de componenten die worden gedeployed;
* Decoupling: voorkomen van propagatie van mutaties over verschillende componenten;
* Event versus request driven: a-synchroon middels het gebruik van queues biedt grote voordelen;
* Service discovery: het automatisch vinden van services door clients;
* Test strategie: End-to-end testing versus unit testing;
* Non functional requirements zoals performance: welke performance wordt verwacht en hoe wordt die gerealiseerd;
* Load balancing, Redundancy en Failure handling: hoe wordt dit bereikt;
* Logging: de richtlijnen ten behoeve van monitoring en metrics zijn niet besproken;
* BI: het is onduidelijk hoe dit component wordt gevuld en ontsloten;
* IAM: het IAM component is genoemd, niet gesproken.
* Azure: hoe de deployment van componenten naar Azure wordt uitgevoerd en welke componenten van Azure worden gebruikt, is niet duidelijk geworden.

# Deployment model

Het doel is een werkende applicatie. Dit betekent dat een deployment model, gevat in een overzichtelijk diagram niet mag ontbreken. Functionele componenten landen in zo’n diagram op de infrastructuur. Het geeft houvast voor de ontwikkeling van de noodzakelijke deployment scripts in de CI/CD pipeline. Een deployment architectuur geeft inzicht in de impact van een wijziging en structuur en bij het beoordeling en oplossen van een outage.

Er is een CI/CD pipelines gerealiseerd. De pipeline start bij iedere merge naar master, maar stopt helaas na deployment in de zogenaamde staging environment. Het testen van de pipeline is van groot belang: er worden wijzigingen in productie mee aangebracht. De aanbeveling is het ontbrekende deel zo snel mogelijk toe te voegen en veel en vaak te testen.

# High availability?

We hebben kort gesproken over de eisen die gesteld worden aan de beschikbaarheid van de Connecthing systemen. Er is daarbij naar voren gekomen dat er 2 soorten systemen worden ontwikkeld:

* Nutselect applicatie: 08:00 - 22:00 uur op doordeweekse dagen;
* Vanzelf applicatie: 00:00 - 24:00, 24 x 7 x 365.

De systemen zullen die requirements moeten realiseren met een beschikbaarheidspercentage van bijvoorbeeld 99,2%. Met de juiste getallen kan de architectuur daarop worden ontworpen.

Beide systemen delen componenten. Dat betekent dat alle componenten geschikt moeten zijn voor deployment zonder down-time. Ik kan niet beoordelen of de set-up van alle componenten dit ondersteunt, waarmee het een risico vormt voor genoemde requirements.

De genomen maatregelen ten aanzien van disaster recovery zijn ook niet besproken.

# Scope

Ten aanzien van de scope voor versie 1.5 wordt opgemerkt dat die niet glashelder is. Een uitspraak doen over de haalbaarheid van versie 1.5 lijkt daarom niet goed onmogelijk.

Het ontbreken van duidelijke scope leidt mogelijk ook tot het ontbreken van noodzakelijke architectuur en software design. Dit introduceert vanzelf technical debt, omdat short-cuts moeten worden genomen om deadline(s) te halen.

# Operatie

Dit onderdeel is niet besproken.

# Hybride

Door de dead-line voor oplevering van het nieuwe platform, inclusief nog niet bekende extra functionaliteit, moeten de keuzes die in het verleden zijn gemaakt (al dan niet terecht), tegen het licht worden gehouden. Over de inschattingen die gemaakt zijn voor het realiseren van het platform lijkt consensus te bestaan over de haalbaarheid. Deze wordt alom als zeer onwaarschijnlijk ervaren. Dit heeft een demotiverende werking tot gevolg.

Advies is met ontwikkelaars (in ieder geval de senior teamleden) alle mogelijke opties te bedenken en te bespreken en samen met hen oplossingen te ontwerpen. De creativiteit van individuen wordt hiermee maximaal benut maar ook de motivatie voor de teamleden om zich te commiteren en het gedefinieerde doel te bereiken via de voorgestelde route.

Het team geeft aan dat als de .NET strategie wordt gehandhaafd, versie 1.5 in Q3 wordt opgeleverd. Als daarentegen een hybride strategie wordt gekozen, de oplevering van versie 1.0 inclusief (enkele) pijnpunten mogelijk zou moeten zijn per einde Q2. Een volledig oplevering in .NET zal dan langer duren en pas in Q4 kunnen plaatsvinden. Dat lijkt mij voldoende reden een nieuwe afweging te maken en samen met (senior) teamleden een strategieen te ontwikkelen, waaruit het management een keuze kan maken.