Messaging

casestudie

groep 3

2021

Inhoud

[The case 2](#_Toc68169983)

[Onderzoeksvragen 3](#_Toc68169984)

[Strategie 3](#_Toc68169985)

[resultaten 4](#_Toc68169986)

[Welke messaging systemen zijn er beschikbaar voor High Sky? Wat zijn de voor- en nadelen hiervan? 4](#_Toc68169987)

[Messaging Queue 4](#_Toc68169988)

[Socket/Websocket 4](#_Toc68169989)

[Hoe kan er voor gezorgd worden dat data niet verloren gaat wanneer er iets fout gaat in de database? 5](#_Toc68169990)

[Hoe kan er voor gezorgd worden dat data niet verloren gaat bij het verlies van services? 5](#_Toc68169991)

[Hoe kan ervoor gezorgd worden dat het systeem makkelijk integreerbaar is voor nieuwe partijen? 6](#_Toc68169992)

[Waar moet organisatie High Sky op zijn voorbereid? 6](#_Toc68169993)

[Architectuur voorstel 7](#_Toc68169994)

[Conclusie 8](#_Toc68169995)

[Bronnen 9](#_Toc68169996)

# The case

Author: Leon Schrijvers

Date: January 27, 2021

The research and development agency 'High Sky' is working for High Tech companies in the Brainport region. The company has a sound reputation of transforming research prototypes, MVPs, walking skeletons and PoCs into production systems. Recently it was approached to architect a cutting-edge AI research project.

The research project involves over 10 different parties, including companies, non-profit organisations and research institutions. The theme of the project is to create a mixed reality in which AI avatars help to train real people in dangerous or stressful situations. Participants are closely monitored by IoT devices and partly by the virtual events in the mixed reality. The system anticipates changes in the mood, stress levels of the participant and reacts accordingly in real time to provide realistic experiences.

Until now, the 15 parties that cooperate on a same project could not agree on the common architecture, technology stack and all have their own specific requirements. This is mainly caused by the fact that involved parties have already written parts of the system in different programming languages, like C/C++/Python, a subset of .NET (Unity engine) and own twist on C++ (Unreal engine). Note that none of the software is web based.

What was agreed is that each party continues to develop their own technology to keep investment low for each party. The development releases will not be synchronised among parties as it would disturb core business processes of each participant.

To goal for 'High Sky' is to design a *system of systems*, in which all the systems are stitched together into one reliable and real-time solution. The resulting design should support the following capabilities:

* Realtime data processing of high volume IoT sensor data
* Data is not *broadcasted* to all involved parties, but parties only receive data they are interested in
* Data may be analysed directly, but also needs to be archived for future analysis and trouble shooting
* Robust storage of data (no data should get lost in case of a failure)
* Graceful degradation of services
* Development platform independent
* Open to future parties that wants to integrate with the system.

# Onderzoeksvragen

Voor dat er een ontwerp gemaakt kan worden voor een architectuur. Moeten we eerst onderzoek doen naar relevante informatie over het onderwerp messaging. Om het onderzoek te leiden zijn er vragen opgesteld met als hoofdvraag:

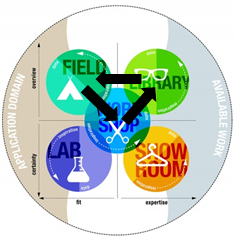
*“Hoe kunnen verschillende platform onafhankelijke systemen op een real-time en robuuste manier aan elkaar gekoppeld worden door middel van messaging systemen?”*

Omdat dit een brede vraag is, zijn er deelvragen opgesteld. Deze deelvragen hebben als doel meer duidelijkheid te creëren over een specifiek onderdeel met betrekking tot de hoofdvraag. De deel vragen zijn:

* Welke messaging systemen zijn er beschikbaar voor High Sky? Wat zijn de voor- en nadelen hiervan?
* Hoe kan ervoor gezorgd worden dat data niet verloren gaat wanneer er iets fout gaat in de database?
* Hoe kan ervoor gezorgd worden dat het systeem makkelijk integreerbaar is voor nieuwe partijen?
* Waar moet organisatie High Sky op zijn voorbereid?

# Strategie

Het verrichten van het onderzoek zal gebeuren met behulp van het DOT Research framework. Voor het gebruik van het framework is het van belang dat er verschillende methodes worden gebruikt. Deze methodes komen uit verschillende strategieën en worden toegepast om je onderzoek te valideren.



Om de onderzoeksvraag te beantwoorden hebben wij gekozen voor de volgende strategieën met daarbij de volgende methodes.

|  |  |  |
| --- | --- | --- |
| **Field** | **Library** | **Workshop** |
| Focus group | Community research | Prototyping |
|  | Available product analysis | IT architecture sketching |

In de groep zijn we begonnen met focus group. Hierbij hebben we geïnventariseerd met welke technieken de groepsleden al in aanmerking zijn gekomen. Vervolgens hebben we uit de vragen die hieruit ontstonden community research gedaan en een available product analysis gedaan om te zien wat er nog meer op de markt is. In ons groepsproject en individuele projecten hebben we hier ook prototypes gemaakt. Zo heeft het merendeel van de groep een werkende verbinding opgesteld met Redis maar ook met Kafka en RabbitMQ. Daarnaast zijn we in een eerder semester in aanraking gekomen met (web)sockets.

# resultaten

De resultaten van de deelvragen zullen in dit hoofdstuk worden beschreven. Aan het einde van dit hoofdstuk wordt er een architectuur voorstel gedaan die een oplossing biedt voor het probleem. De resultaten van dit hoofdstuk zullen worden meegenomen in de conclusie van het onderzoek.

## Welke messaging systemen zijn er beschikbaar voor High Sky? Wat zijn de voor- en nadelen hiervan?

Omdat High Sky op zoek is naar een real-time koppeling tussen de verschillende systemen wordt de zoektocht al een stuk nauwer. Onderstaand worden twee typen messaging systemen kort uitgewerkt en vervolgens met elkaar vergeleken.

### Messaging Queue

Een Messaging Queue (MQ) is zoals de naam al zegt een wachtrij voor berichten. Verschillende systemen communiceren met elkaar door een bericht in de queue te plaatsen (sender). Dit bericht wordt vervolgens gebroadcast naar andere aangesloten systemen of dit bericht kan uitgelezen worden door één of meerdere specifieke systemen (receiver). Een MQ kent een aantal grote voordelen die erg van belang zijn voor High Sky:

* De gekoppelde systemen hoeven vrijwel niks van elkaar te weten. Hierdoor is het erg eenvoudig om verschillende (platform onafhankelijke) systemen aan deze koppeling toe te voegen.
* De MQ dient als een tijdelijke opslag voor niet bezorgde berichten. Hierdoor hoeven de sender en receiver systemen dus niet tegelijkertijd aan de MQ gekoppeld te zijn. Dit zorgt voor een robuuste koppeling.

Een aantal bekende voorbeelden van MQ-systemen zijn Kafka, Redis en RabitMQ.

### Socket/Websocket

In verhouding met een MQ zijn sockets of websockets een veel meer ‘low-level’ oplossing. Een dergelijke connectie biedt full-duplex communicatie tussen de twee gekoppelde systemen. Middels deze koppeling kunnen de twee systemen real-time met elkaar communiceren. Echter, deze systemen moeten daarvoor wel van elkaar weten en beiden online zijn. Om slechts een paar systemen met elkaar te koppelen is dit een erg geschikte oplossing, maar naarmate er meer systemen worden gekoppeld, stijgt de complexiteit explosief.

|  |  |  |
| --- | --- | --- |
|  | Messaging Queue | Socket/websocket |
| Robuustheid | 5/5 | 1/5 |
| Schaalbaarheid | 5/5 | 3/5 |
| Snelheid | 4/5 | 4/5 |
| Platform independent | 4/5 | 5/5 |
| Open voor toekomstige integraties | 5/5 | 1/5 |

## Hoe kan er voor gezorgd worden dat data niet verloren gaat wanneer er iets fout gaat in de database?

Met de hoeveelheid data dat er wordt opgeslagen bij High Sky is het van belang dat er een goede implementatie voor een database wordt gekozen. Een goede implementatie houdt in dat de data niet verloren gaat wanneer de database een keer een storing heeft. Om dit op te lossen kan je denken aan het volgende.

* Maak minimaal een copy van je data en zet dit op een los systeem. Dit systeem heeft geen interactie met het systeem waar de oorspronkelijk van is.
* Zorg voor automatische controle van de backup data om te kijken of deze niet corrupt, incorrect of invalid is.
* Zorg ervoor dat je database een ononderbroken stroomvoorziening heeft voor het geval dat er een stroomuitval is. Hierdoor kan je snel je database normaal uitzetten en heb je geen data verlies.

## Hoe kan er voor gezorgd worden dat data niet verloren gaat bij het verlies van services?

In grootschalige systemen waar applicaties met elkaar communiceren is er kans dat een service binnen het systeem kan uitvallen door mogelijk hardware fouten of bugs in code. Als dit overkomt is er kans op data verlies die nog in het systeem zit.

In grootschalige systemen wordt er vaak gebruikt gemaakt van event logging. De meeste messaging systemen die gebruikt kunnen worden hebben deze functie zelf. Zij houden events bij die afgehandeld moeten worden in het systeem. Deze events hebben een status of zij al afgehandeld zijn en worden in een aparte database vastgehouden.

Kafka heeft bijvoorbeeld de functionaliteit van events vasthouden. Kafka slaat events op in zijn bestanden, op basis van deze bestanden zou een gehele database opnieuw gevuld kunnen worden. Kafka houdt alle data vast die gedaan uitgewisseld wordt over Kafka. Al valt Kafka weg in het systeem wordt alles bij een restart opgelost.

Redis is te vergelijken met Kafka als het gaat over events vasthouden. Bij Redis worden de events alleen opgeslagen in een memory database. Als Redis zou weg vallen ben je dus de events kwijt.

## Hoe kan ervoor gezorgd worden dat het systeem makkelijk integreerbaar is voor nieuwe partijen?

Het systeem zal data afhandelen met een message queue, door dit te implementeren is het al vrij makkelijk voor een nieuwe partij om het systeem in gebruik te nemen. In het geval dat de nieuwe partij hun API’s via de message queue wil laten praten met andere API’s/systemen, is connectie maken met de message queue het enige wat ze hoeven doen.

Dit zorgt ervoor dat zelfs zeer complexe systemen vrij eenvoudig kunnen worden geïntegreerd binnen het systeem. Hiernaast zorgt de manier van aanroepen via message queue er ook nog eens voor dat routes niet ‘-hard-coded’ hoeven te zijn en makkelijk te vervangen zijn met andere functies op dezelfde route (doormiddel van topics)

## Waar moet organisatie High Sky op zijn voorbereid?

De requirements die opgesteld zijn voor de architectuur nemen natuurlijk ook een paar dingen mee waar High Sky zich bewust van moet zijn. Deze worden hier opgesomd:

* Meer data om op te slaan door meerdere backups.
* Hogere ontwikkelingskosten voor de eerste opzet voor lagere onderhoudskosten later in het project.
* Meer tijd nodig voor documentatie om andere bedrijven makkelijk te laten integreren.
* Complexe architectuur.

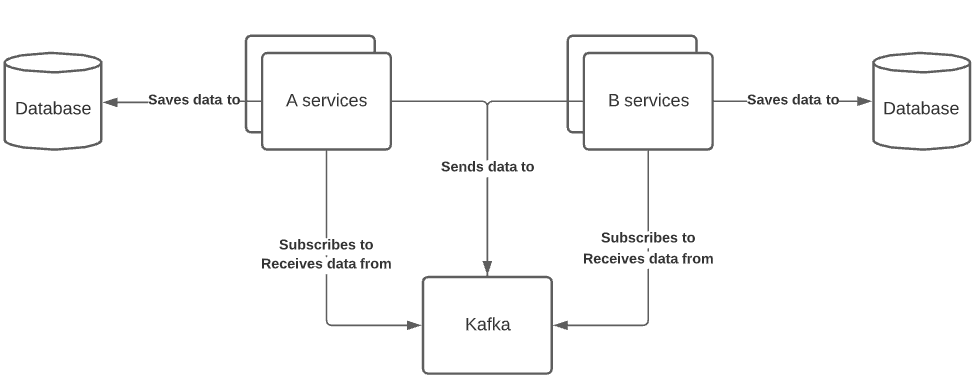
Hoewel dit grote gevolgen kan hebbe, als het bedrijf ertegen is voorbereid zouden geen van deze dingen ervoor moeten zorgen dat de applicatie niet goed wordt ontwikkeld/onderhouden.

## Architectuur voorstel

Het voorstel die wij maken voor High Sky is om via Kafka te communiceren. Kafka houdt events vast en geeft zekerheid dat events worden afgehandeld.

Voor High Sky zijn de systemen in verschillende talen geschreven en worden ze op verschillende plekken gehost. Kafka kan hiervoor als een centraal punt handelen waar communicatie met services zal plaatsvinden. Kafka kan ook gebruikt worden als event geschiedenis opslag/overzicht.

Het onderstaande diagram geeft een overzicht van hoe Kafka binnen het bestaande systeem gekoppeld kan worden.



# Conclusie

Om de verschillende systemen van High Sky op een real-time en robuuste manier met elkaar te kunnen koppelen hebben wij naar verschillende oplossingen gekeken en deze met elkaar vergeleken. Omdat High Sky op zoek is naar een schaalbaar systeem, waar toekomstige systemen eenvoudig aan kunnen integreren, zijn we al snel tot de conclusie gekomen dat een Messaging Queue de geschikte oplossing is.

Wij hebben vervolgens de keuze gemaakt voor Kafka om dit een extra stuk robuustheid oplevert in de vorm van opslag voor de events in de queue. Dit betekent dat als Kafka tijdelijk wegvalt, de events niet verloren gaan.

# Bronnen

Bell, A. (2017, 3 november). *How to Prevent Data Loss: 7 Ways to Safeguard Your Company*. Solid State Systems LLC. http://solidsystemsllc.com/prevent-data-loss/

fdreger. (2021, 27 feburari). *Message queues vs sockets.* StackOverflow accepted answer.  
Geraadpleegd op 1 april 2021 van: <https://stackoverflow.com/questions/10668028/message-queues-vs-sockets/10668139#:~:text=Socket%20allows%20connection%20between%20clients,in%20a%20message%2Ddriven%20system>.