Integration patterns for microservices architectures: The good, bad and ugly

# Monolithic

Alles in één databank.

In één aanvraag kan alles gebeuren.

Rollback transaction is gemakkelijk.

# Microservice

Definitie:

Bij verandering moeten andere services genotifieerd worden.

Het behouden van consistency is belangrijk.

# Database sharing

## Aparte service om de databank in sync te houden

Elke microservice heeft eigen databank.

Door aparte microservice te gebruiken om databanken in sync te houden. Dit zorgt ervoor dat er meer verbindingen liggen tussen microservices.

Geen goed idee om deze methode te gebruiken.

Microservices mogen geen databank sharen.

## De microservice update andere microservices hun databank

Gebeurt er iets in de microservice dan meldt de microservice die meteen aan de anderen.

# REST integration

Rest endpoint maken van microservices zodat anderen die kunnen gebruiken om meldingen te sturen bij veranderingen.

Implementatie blijft geheim

Als er veel microservices zijn die communiceren met elkaar, dan moet er maar één microservice zijn die traag is. Dit heeft gevolgen voor de gehele architectuur.

Problemen:

* Temporal coupling  
  Er is een service niet beschikbaar en er wordt een request naar hem gestuurd. Te afhankelijk
* Latency  
  Bij het sturen van een request, komt de andere in een bezet-toestand waardoor er vertraging kan oplopen. Er kan pas verder gewerkt worden wanneer de service uit de bezet-toestand komt.

# Messaging

Message broker

Bij verandering wordt er gestuurd naar de message broker.

Andere services luisteren naar de message broker.

De melding wordt door elke instantie op zijn tempo verwerkt.

Er kan geen response gestuurd worden.

Belangrijke concepten:

* De berichten worden opgeslagen.
* De microservices weten niet welke er luisteren
* Events worden op een asynchrone manier verwerkt.
* Publish-subscribe patroon

ETL processen???

Microservice integration patterns, sap

De integratie patronen kunnen in categorieën gestoken worden:

* UI
* Messaging
* API

# UI

Maak gebruik van minimale koppeling. Zorg dat het niet te ingewikkeld wordt.

# Messaging integration

Publish-subscribe: topic-based publish-subscribe

Asynchronous communicatie

Guaranteed delivery

Extensible

No service discovery

# API integration

Om microservices met elkaar te connecteren.

# Anti-pattern: database integration

Meerdere microservices kunnen een centrale databank aanspreken.

Integration patterns for microservices architectures

Bla bla

How integration patterns impact your microservices architecture, Yogi Aradhye

# Database integration

Services lezen en schrijven data naar een centrale databank.

Simpel.

# Synchronous API calls

Communiceren aan de hand van API’s.

Toegang tot een ander zijn data: via api

Implementatie details worden verborgen.

# ETL (Extract, transform and load)

Synchroon data via de achtergrond met een schema.

Enkel ETL processen moeten aan de databank kunnen.

Asynchroon. Zonder callback

Implementatie details worden verborgen

Gebruikers ondervinden geen invloed

# Messaging

Berichten uitwisselen via commando’s of integratie events.

Publish-subscribe pattern.

Good decoupling

De publisher ondervindt geen last van de subscribers hun afhandeling

De consistency kan wel problemen ondervinden of verminderen.

Categorieën:

* Queueing solution  
  live data  
  opslaan van bericht
* Streaming solution  
  Berichten opslaan op een stream.   
  Veel configuratie

# Parting thoughts

Fail scenarios hebben

Netflix: gebruikt messaging en synchronous api bij falen