Microservices

Handling Changes

Hönnun og smíði hugbúnaðar

Haust 2022

Structural vs Semantic Breaking Changes

- Structural
  - Breyting á strúktúr á gögnum
  - Auðveldar að finna en semantic
  - T.d. að fjarlægja field úr dto
  - Hægt að finna með schemas og Consumer Driven Contracts

- Semantic
  - Meira vandamál en structural
  - Breyting á hegðun / virkni
  - T.d. að breyta *allowance* field frá nettó í brúttó
  - Hægt að finna með Consumer Driven Contracts

Avoiding Breaking Changes

- Við breytingar viljum við koma í veg fyrir *breaking changes*
- Þ.e.a.s viljum ekki að client / consumer brotni við breytingar
- Nokkrar leiðir til að hjálpa við þetta

Avoiding Breaking Changes

Expansion Changes

- Viljum bæta við interface en ekki breyta því
- Þannig brjótum við ekki tilstandandi samninga
- T.d. bæta við nýju field-i í dto
- T.d. bæta við nýjum endapunkt

Avoiding Breaking Changes
Tolerant Reader

- Consumer ætti að reyna að vera vera robust
- *“be conservative in what you send, be liberal in what you accept”*
- Consumer ætti t.d. ekki að binda sig við gögn sem hann þarf ekki

Avoiding Breaking Changes
The Right Technology

- Viljum velja tækni sem gera breytingar auðveldar
- Flest communication tækni í dag nokkuð góð
  - REST, gRPC, graphQL, Kafka , RabbitMQ Etc.
- Kafka, gRPC sérstaklega gott því þau bjóða upp á explicit schemas
  - Sérstaklega Kafka með Schema versioning virkni
- REST getur haft sína kosti með HATEOS

Avoiding Breaking Changes
Explicit Interface & Schemas

- Explicit Interface / Schema geta hjálpað mikið
- Svipað og hvernig static typing hjálpar vs dynamic typing
- Setur skýrt fram hvað consumer má búast við og hvað er expose-að
- Hjálpar við *Structural breakages* en ekki *Semantic Breakges*
- Explicit Schemas bjóða oft upp á client code generation
  - T.d. gRPC
  - Einhver stuðningur fyri REST með OpenAPI
- Meiri stuðningur í sumri tækni
  - T.d. Í gRPC, Kafka
  - Lítill stuðningur í RabbitMQ og REST (Þó sjá OpenAPI varðandi REST)

Avoiding Breaking Changes

Catch Accidental Breaking Changes Early

- Schemas geta hjálpað okkur að finna *breaking structural changes*
  - T.d. Kafka Schema Registry

- Consumer Driven Tests geta hjálpað okkur að finna *breaking sementic changes*

Managing Breaking Changes

- Þetta er það sem við reynum að forðast
- Við munum einhvern tímann þurfa að gera **incompatible changes**
- Þrjár megin leiðir til að ráða við þetta
  - Lockstep deployment
  - Coexist incompatible microservice versions
  - Emulate the old interface

Managing Breaking Changes
Lockstep Deployment

- Breytum og deploy-um service-um saman
- Leyfir okkur að breyta og deploy-a client saman með ósamþýðanlegu breytingunum
- Spýtir í augun á *independent deployability*
- Getur verið besta leiðin en við **viljum ekki normalize-a Lockstep deployment**

Managing Breaking Changes
Coexist Incompatible Microservice Versions

- Getum haft mismunandi version instance af service keyrandi
- Eldri consumers eru routaðir í gamla version-ið
- Gerir viðahald mjög erfitt
- Þarft flóknar routing reglur
- Eitthvað sem við ættum að **reyna að aldrei gera**
- (þetta á ekki við um blue/green eða canary deployments)

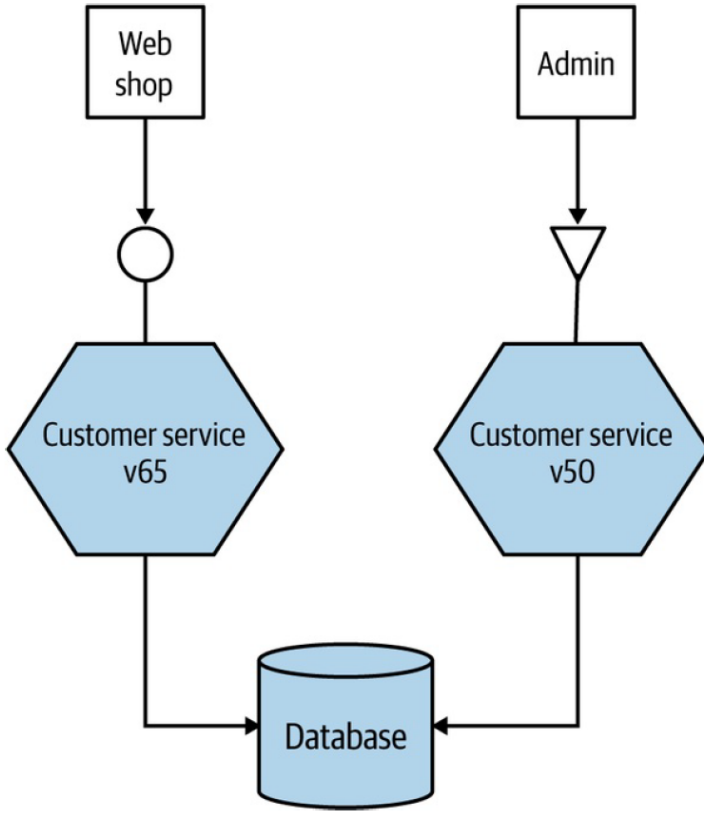

Figure 5-3. Running multiple versions of the same service to support old endpoints

Managing Breaking Changes
Emulate the Old Interface

- Kannski besta leiðin er að **viðahalda version-um af interface-um**
- Getum gert þetta með því að viðráðanlegra með því að endurskrifa gamla interface-ið með köllum í það nýja (**Adapter pattern-ið**)
- Kemur í veg fyrir að lockstep deployment
- Getum breytt consumers hægt og rólega til að nota nýja interface-ið