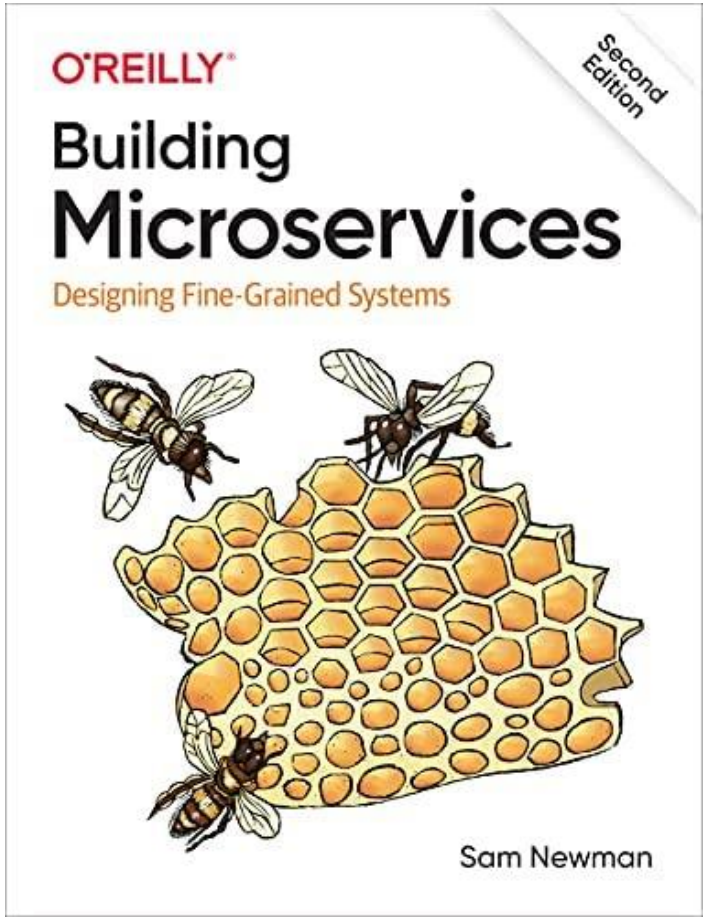
Microservice Architecture

Overview

Hönnun og smíði hugbúnaðar

Haust 2022

Bókin


O'REILLY
Second Edition
Building
Microservices
Designing Fine-Grained Systems
Sam Newman

Hvað eru microservices?

- Mörg *tiltölulega lítil* **decoupled** service
- **Domain Partitioned** Architecture (vertical sliced)
- Hvert service með sitt eigið **bounded context** fyrir ákveðið **domain** með **skír mörk**
- Hafa **stakan tilgang** og **gera einn hlut vel**
- Hvert service **sjálftætt** frá öðrum
- Tala við hvort annað í gegnum **network calls** (REST, Events, GraphQL, gRPC etc.)

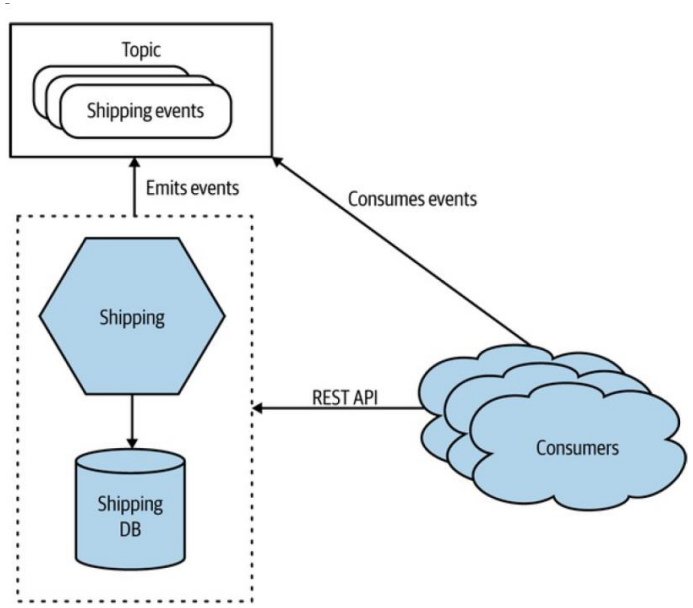


Figure 1-1. A microservice exposing its functionality over a REST API and a topic

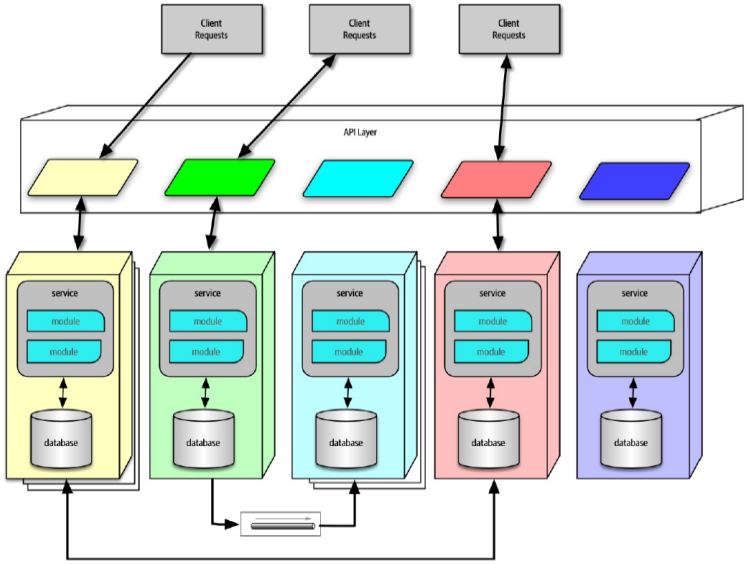
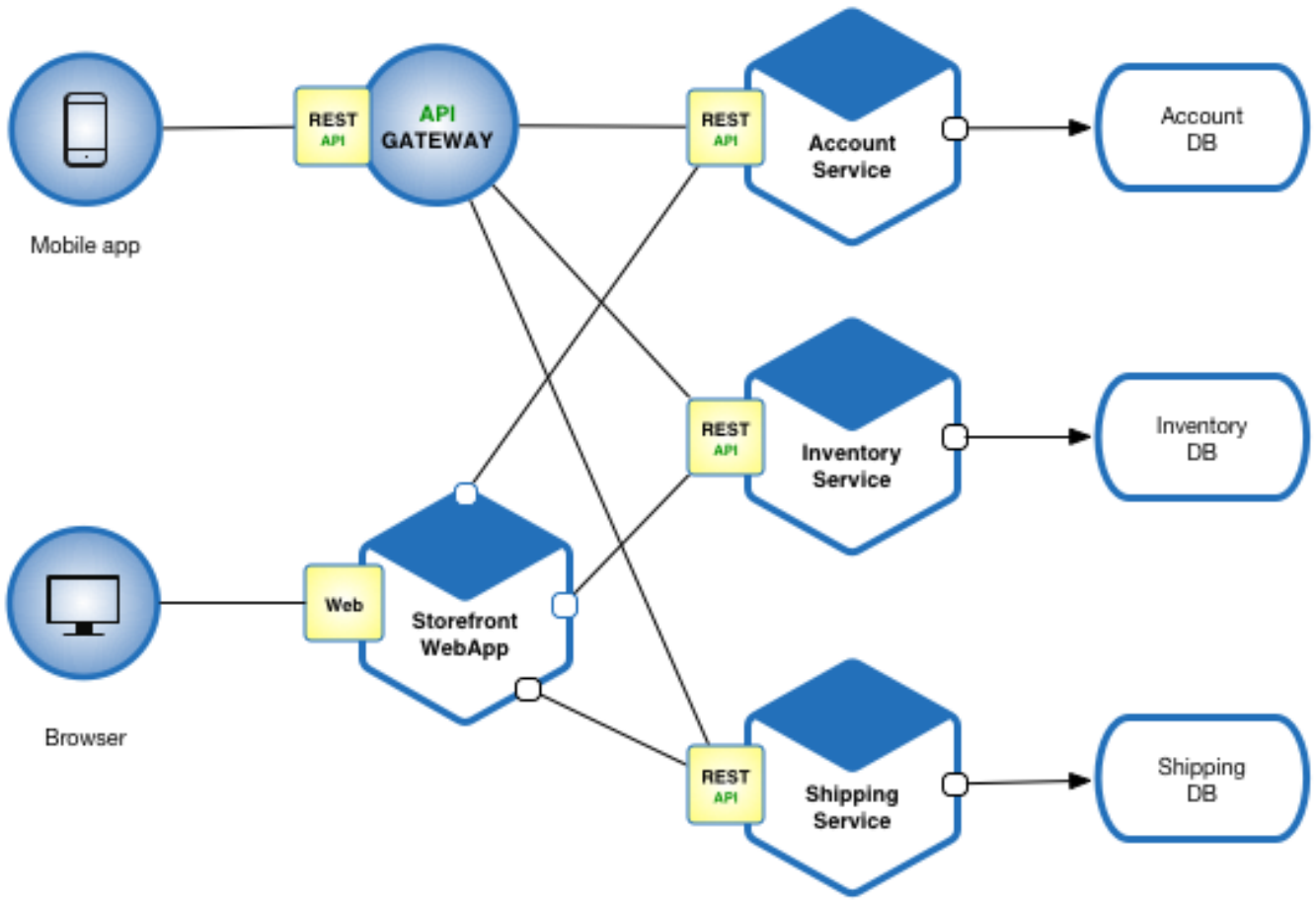
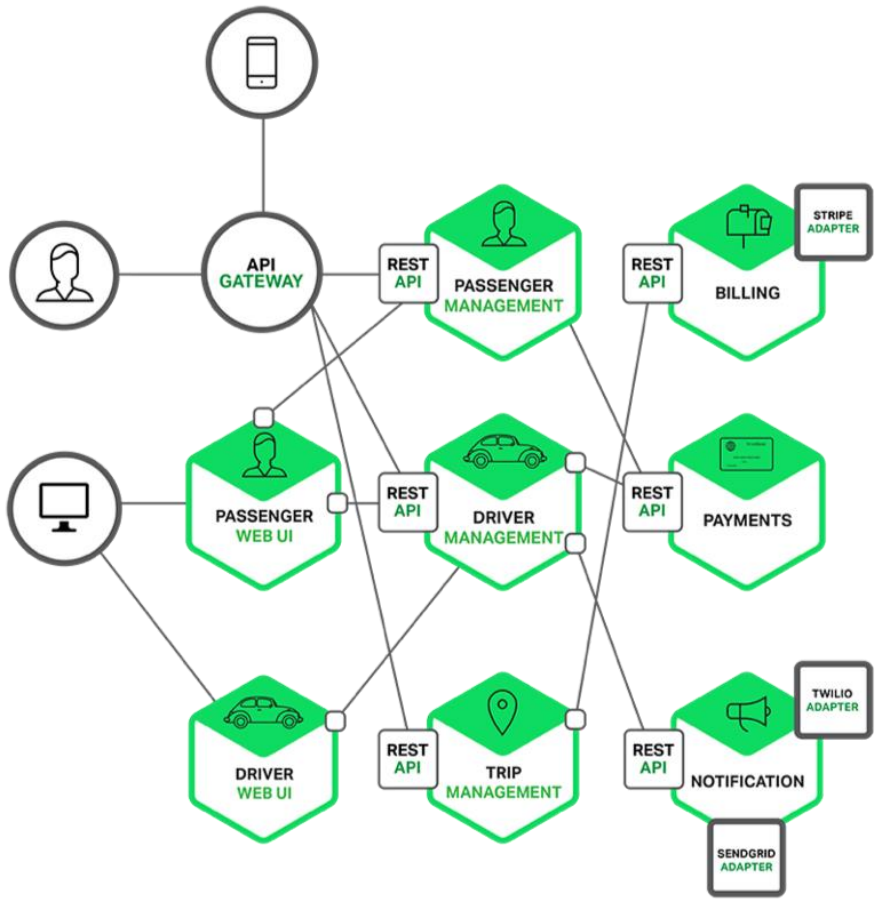


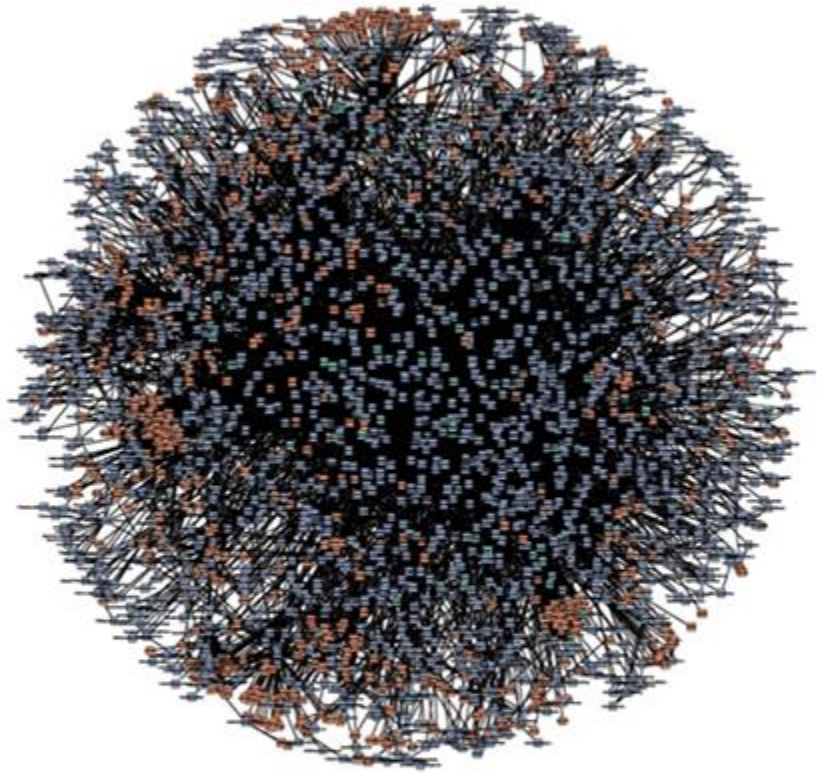
Figure 17-1. The topology of the microservices architecture style

Hvað eru microservices? Frh.

- Hvert service er **independently deployable**
- Hvert service er **technology agnostic**
- Hvert service lifir í **einangruðu umhverfi** (t.d. Docker Container)
- Hjúpa og fela **sína virkni** og **sín gögn** **(**með API og **eigin gagnagrunn**)
- Auðvelt að **breyta einu service án þess að hafa áhrif á önnur** (í fullkomnum heimi)
- Þessi service **mynda saman flóknara kerfi** (*The whole is greater than the sum of its parts*)


Mobile app
REST API
API GATEWAY
REST API
Account Service
Account DB
REST API
Inventory Service
Inventory DB
Browser
Web
Storefront WebApp
REST API
Shipping Service
Shipping DB


API
GATEWAY
REST
API
PASSENGER
MANAGEMENT
REST
API
BILLING
STRIPE
ADAPTER
PASSENGER
WEB UI
REST
API
DRIVER
MANAGEMENT
REST
API
PAYMENTS
DRIVER
WEB UI
REST
API
TRIP
MANAGEMENT
REST
API
NOTIFICATION
TWILIO
ADAPTER
SENDGRID
ADAPTER



amazon.com

NETFLIX

Domain partitioned

- Services eru *Modeled around a business domain*
- *End-to-end slices of business functionality*
- Microservice Architecture er þannig Vertically Sliced
- Tekur innblástur frá Domain Driven Design (DDD)
- Leiðir til skíra marka á milli service-a (*bounded context*)
- Leiðir til
  - flexibility, adaptability, reusability, decoupling, composability, scalability

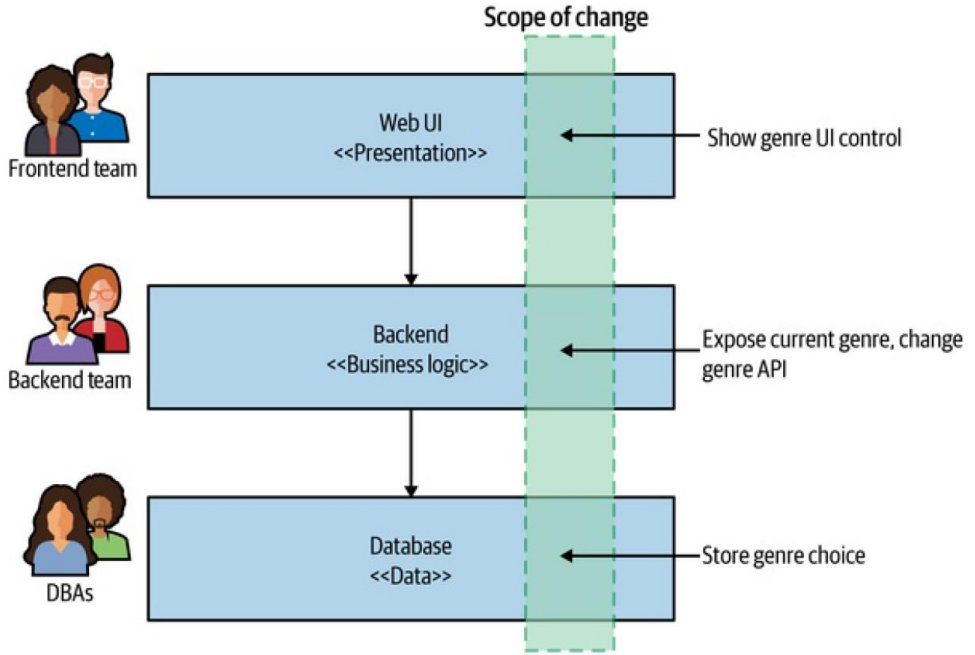


Figure 1-3. Making a change across all three tiers is more involved

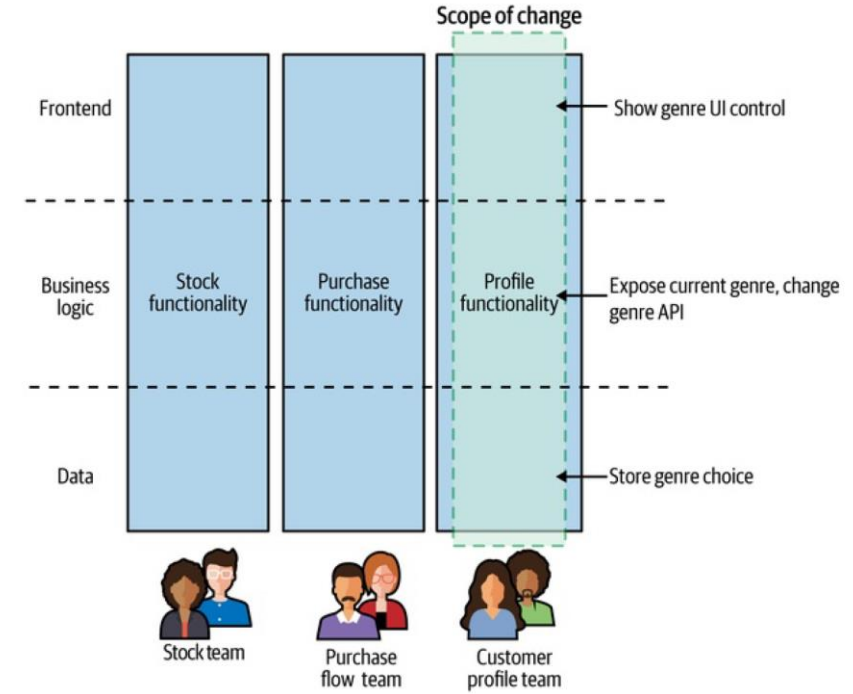


Figure 1-4. The UI is broken apart and is owned by a team that also manages the serverside functionality that supports the UI

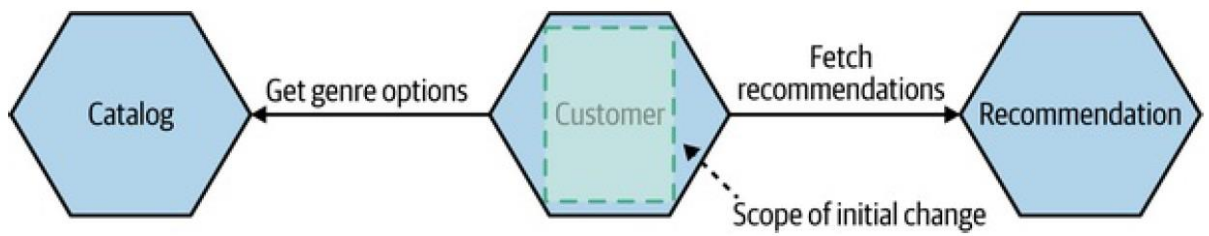


Figure 1-5. A dedicated `Customer` microservice can make it much easier to record the favorite musical genre for a customer

Conway‘s Law

CONWAY’S LAW

Back in the late 1960s, Melvin Conway made an observation that has become known as *Conway’s law*:

> *Organizations which design systems … are constrained to produce designs which are copies of the communication structures of these organizations.*

Paraphrased, this law suggests that when a group of people designs some technical artifact, the communication structures between the people end up replicated in the design. People at all levels of organizations see this law in action, and they sometimes make decisions based on it. For example, it is common for organizations to partition workers based on technical capabilities, which makes sense from a pure organizational sense but hampers collaboration because of artificial separation of common concerns.

A related observation coined by Jonny Leroy of ThoughtWorks is the Inverse Conway Maneuver, which suggests evolving team and organizational structure together to promote the desired architecture.

Blackbox

- Service virka sem blackbox
  - Veist ekki innri virkni
  - Talar við service í gegnum interface (eða með events)

- Service hafa vel skilgreindan API
  - Service ráða hvaða virkni og gögn eru opinberuð
  - *Embrace the concept of information hiding*
  - API gæti verið í formi event schema, REST, gRPC, graphQL etc

- Information Hiding
  - Viljum fela eins mikið og mögulega
  - Viljum expose-a eins litlu og mögulega

- Hjúpa sín gögn
  - Hvert service á og hjúpar sín gögn
  - Hvert service á þ.a.l sitt eigið data store (t.d. SQL gagnagrunn)
  - Data store-ið er aðgengilegt eingum nema service-inu sem á það
  - Ef eitt service vantar gögn frá öðru þá eru þau sótt í gegnum API

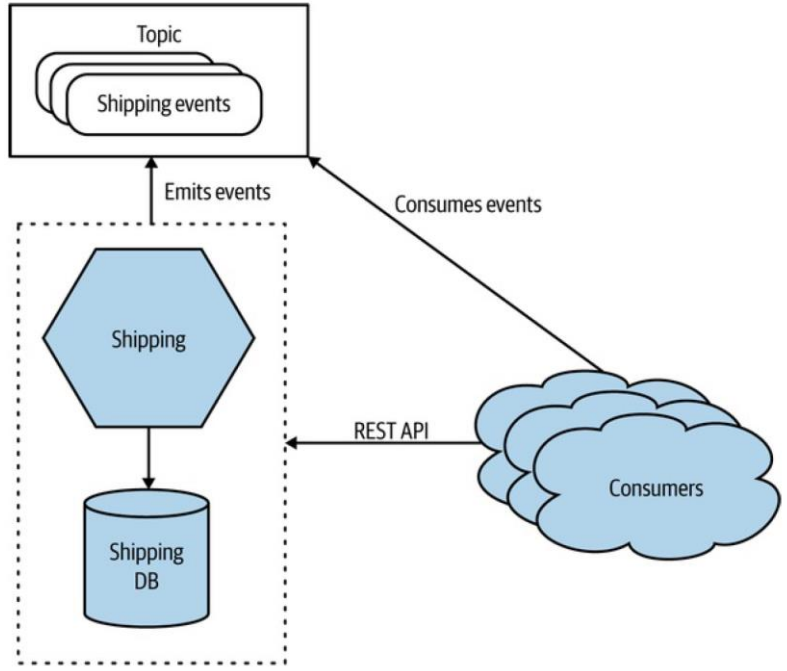
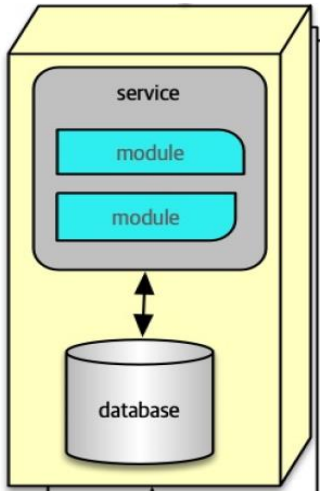


Figure 1-1. A microservice exposing its functionality over a REST API and a topic

Hjúpun Gagna



- Microservices forðast gagna coupling
- Hvert service er eigandi sinna gagna (**á sitt data eigið data store**)
- Service ræður birtingu gagna með API
- Gallar
  - Endurtekt
    - Gætum þurft að hafa kóða / gagna duplication með þessu fyrirkomulagi
  - Transactions
    - Transactions eru erfið í distributed kerfi (ekki sameiginlegur gagnagrunnur)
  - Gagna samræming
    - Getum ekki samræmt gögn á milli service-a með foreign keys
    - Sömu gögn geta verið í mörgum grunnum (og engin trygging á samræmi)
- Kostir
  - Technological Heteroginity
    - Hvert service ræður data store-inu sínu
    - T.d. eitt service með SQL grunn, annað með Document Database, annað með Graph database etc.
  - Decoupling
    - Service vita ekkert um innri framsetningu gagna í öðru service-i
    - Breyting á gögnum / gagna strúktúr / gagnagrunn er falið frá öðrum service-um

TIP

Don’t share databases unless you really need to. And even then do everything you can to avoid it. In my opinion, sharing databases is one of the worst things you can do if you’re trying to achieve independent deployability.

Loose Coupling

- Eitt af helstu hvötum bakvið microservices
- Microservices eiga að vera decoupled frá hvort öðrum
- Breyting í einu service ætti ekki að hafa áhrif á annað
- Okkur tekst þetta með vel skilgreindum APIs
  - Verndar clients frá breytingu á gögnum og virkni
  - Events gefa okkur sérstaklega decoupled services

Cohesion

- *the code that changes together, stays together*
- Skyldur domain kóði á heima saman
- Viljum hafa hátt cohesion innan service
- Viljum bara þurfa að gera breytingar í einu service
- Mikilvægt að finna rétt *service boundary*

The Perils Of Reuse

- Með microservices viljum við fórna DRY fyrir decoupling
- Services ættu að deila eins litlum kóða og gögnum og mögulega
- Viljum frekar endurtaka okkur
  - Meikar oft sens að endurtakta kóða
  - Sama domain model getur verið í mörgum services
  - Í microservice architecture er oft viðurkennt að hafa endurtekt af gögnum
  - Getum t.d. deilt kóða með templates
- Getum splittað endurtekt í sér service

The Perils Of Reuse – Shared Libraries

- Missum independently deployability
- Allir sem nota library-ið coupled
- Libraries geta orðið over-generalized
- Erfitt að gera breytingar á library-inu

- Libraries ættu **aldrei** að snúast um domain þætti
- Á ekki við um 3rd party libraries
- Eigandi virkninnar óljós

The Perils Of Reuse - Databases

- Shared Databases mynda coupling
- Missum independent deployability
- Missum gagna hjúpun
- Erfitt að gera breytingar á gögnum
- Breyting á gögnum og strúktúr leiðir til breytinga í coupled services
- Leiðir til over generalized gögn
- Allir sem nota gögnin coupled saman
- Getur leitt til performance vandamála
- Óljóst hver uppspretta og eigandi gagnana er

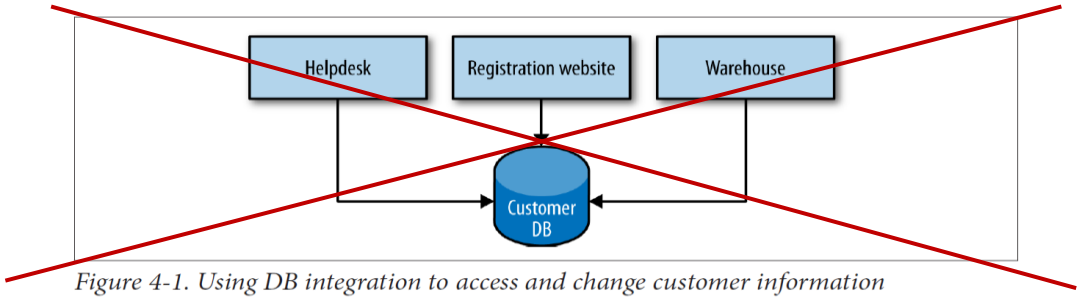


Figure 4-1. Using DB integration to access and change customer information

Cross-Cutting Concerns

- Sumt meikar sens að deila á milli microservices með t.d. libraries
  - T.d. Logging, Monitoring, Communication etc.
- Þetta snýr að hlutum sem þarf að vera staðlað fyrir allan microservice flotann
- Þessir þættir ættu að breytast sjaldan
- Hefur sömu copuling vandamál og endurnotkun á libraries

Independent Deployability

- Við viljum geta breytt og deploy-að services í einangrun
- Viljum geta deploy-að einu microservice-i í einu
- Viljum forðast step-lock deployment eða versioning
- Þurfum að forðast backward-incompatible breytingar
- Krefst þess að services eru loosely coupled
  - Krefst hjúpun á gögnum
  - Vel skilgreint API
  - Information Hiding
- Krefst þess að service boundary-in okkar eru rétt
  - Breytingar sem geta verið deployed í einangrun þurfa að vera í sama service
  - Service mega ekki vera of lítil
- Auðveldara sagt en gert

Independent Deployability

TIP

If you take only one thing from this book and from the concept of microservices in general, it should be this: ensure that you embrace the concept of independent deployability of your microservices. Get into the habit of deploying and releasing changes to a single microservice into production without having to deploy anything else. From this, many good things will follow.

Hversu micro er micro?

- Afstætt
- Enginn harðsett regla
- *“A microservice should be as big as my head”*
- Viljum ekki fara of smátt
  - Breytingar ættu að vera hjúpaðar innan microservice (Independent deployability)
  - Dýrt að samræma breytingar á milli service-a
- Ítrun
  - Finnum sjaldan rétta stærð við fyrstu ítrun
  - Tekur tíma að finna rétt domain boundary og stærð
- Betra að byrja stærra og síðan brjóta niður eftir þörfum

Hversu micro er micro? Þumalputta reglur

- Services eiga að fanga ákveðið domain
  - Sum domain eru stærri en önnu
- Tilgangur ætti að vera skír
- Viljum forðast flókin samskipti á milli service-a
  - Mögulega betra að hafa sem eitt service
- Viljum forðast distributed transactions
  - Distributed transactions eru erfið
  - Transactions eru merki um að þetta eigi að vera eitt service

Hversu micro er micro? Frh.

“Think of adopting microservices as less like flipping a switch, and more like turning a dial. As you turn up the dial, and you have more microservices, you have increased flexibility. But you likely ramp up the pain points too. This is yet another reason I strongly advocate incremental adoption of microservices. By turning up the dial gradually, you are better able to assess the impact as you go, and to stop if required.”

Technology Agnostic

Hvert service getur haft mismunandi tech stack

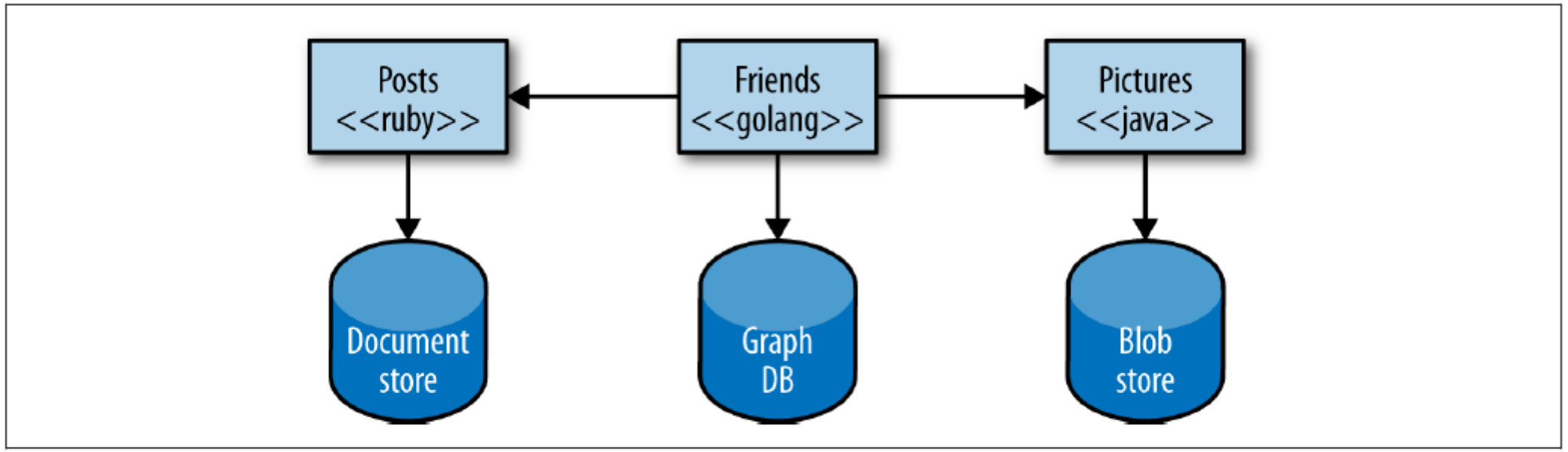


Figure 1-1. Microservices can allow you to more easily embrace different technologies

Communication Methods

- Microservices tala við hvort annað í gegnum network calls

- Margar leiðir / tækni til
  - REST, gRPC, GraphQL, Events

Enabling Technology

- Mikið af nýrri tækni til að aðstoða við microservices
- Bara af því tækni er til þýðir ekki að þú þurfir að nota hana
  - YAGNI
  - KISS
  - Við eigum til með að overkill-a
  - Fyrst sjá vandamál og síðan tækni sem getur hjálpað
- Það að vera architect krefst technology breadth

Enabling Technology
Log Aggregation & Distributed Tracing

- Mjög mikilvægt í distributed architecture

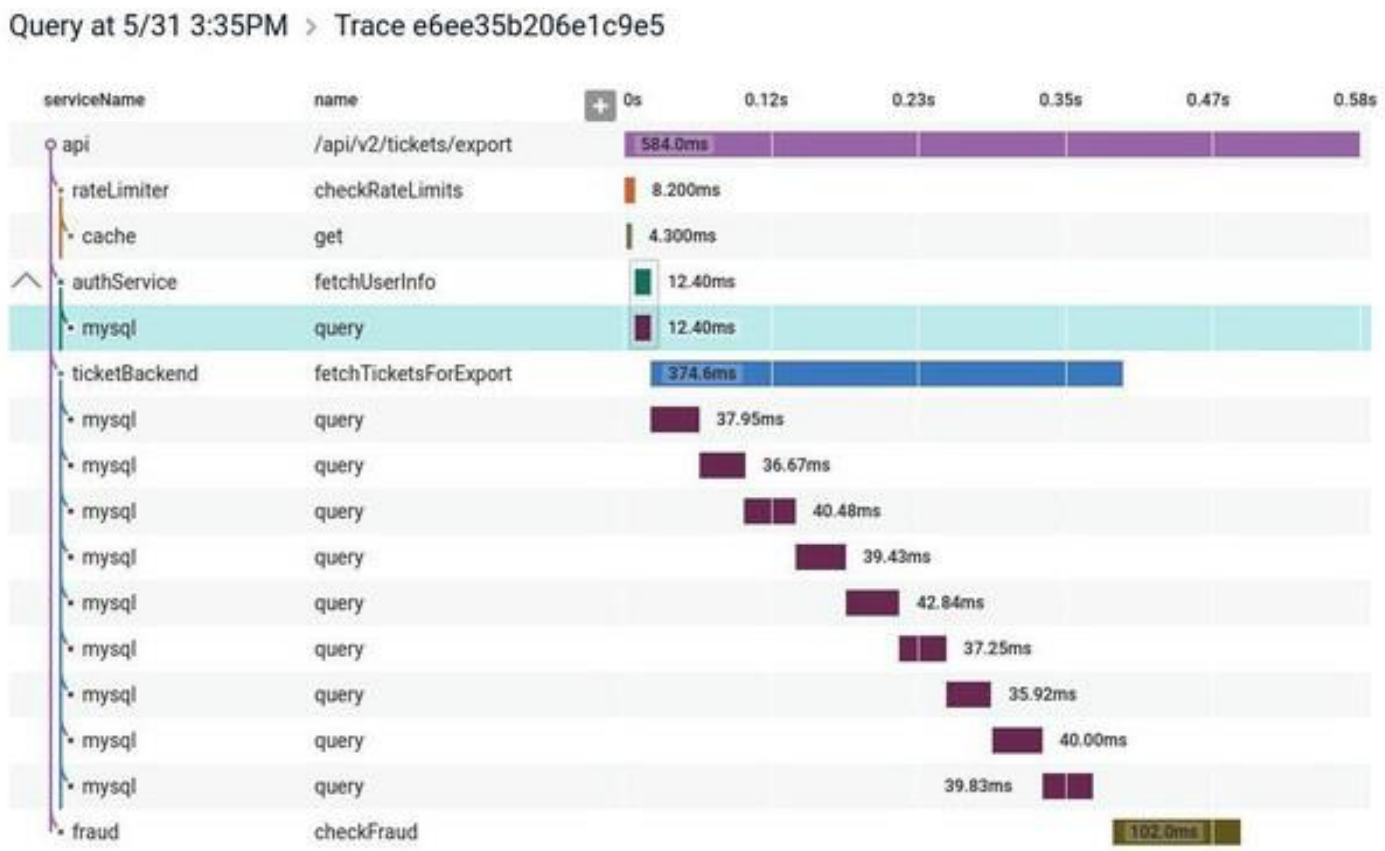
- Þurfum að geta tengt requests, events og workflows á milli service-a

- Getum gert það með því að log-a með trace id
  - Allt sem er log-að innan sama workflows fær sama trace id
  - Loggum t.d. requests og events til að tengja saman flæði
  - Líka kallað: *correlation id, request id*

- Mikið um að velja
  - Datadog
  - Jaeger
  - Honeycomb
  - Lightstep
  - …

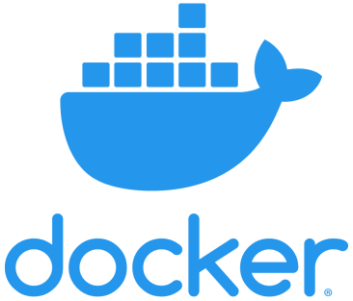
Enabling Technology

Log Aggregation & Distributed Tracing frh



Enabling Technology

Docker & Containers



- Viljum keyra microservices í einangrun
- Viljum geta stjórnað resources sem eru í boði fyrir service
- Viljum að service-ið okkar keyri rétt og eins á öllum umhverfum
- Viljum að DevOps og deployment ferli er auðvelt
- **Docker getur hjálpað okkur við akkurat þessa hluti**

Enabling Technology

Skýjaþjónustur

- Aðkoma skýjaþjónusta gerði microservice architecture mögulegri
- Bjóða upp á haf af deployment, PaaS, serverless og managed services lausnum
- Þrír risarnir
  - AWS
  - Google Cloud
  - Azure

Enabling Technology

Message Queues

- Ein leið fyrir service samskipti er *message queues*

- Með Message Queues senda service *message / event* á queue sem annað / önnur service vinna úr

- Dæmi um tækni
  - RabbitMQ
  - AWS SQS
  - Azure Queue Storage
  - Pulsar
  - (Kafka, í raun ekki message queue en getur þjónað sama tilgangi)

RabbitMQ

Enabling Technology

Event Streaming

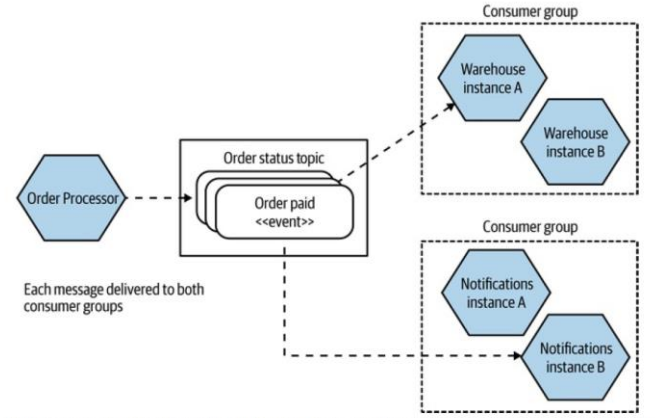


Figure 5-2. Topics allow for multiple subscribers to receive the same messages, which is useful for event broadcast

- Ein leið fyrir service samskipti er *Event driven architecture*
- Með Event streaming publish-ar eitt service event-i um að einhver atburður hefur átt sér stað
- Eitt eða mörg service consume-a síðan þennan event straum og bregðast við þessum atburðum
- Dæmi um tækni
  - Kafka
  - Pulsar
  - AWS Kinesis
  - Azure Event Hubs
  - (RabbitMQ í raun ekki event streaming platform en getur þjónað sama tilgangi)

Enabling Technology

PaaS

- Margar skýjaþjónustur bjóða upp á *highly abstracted* platform til að deploy-a og manage-a service-in okkar.
- Þessar þjónustur kallast Platform as a Service (Paas)
- Með þeim missum við mikið af undirliggjandi stjórn en mikið management overhead er séð um fyrir okkur.
- Dæmi
  - Heroku
  - AWS Beanstalk
  - Google App Engine
  - Azure App Service

Enabling Technology
FaaS

- Við getum deploy-að microservice-inu okkar sem eitt eða fleir föll
- Þessi lausn er keyrð og managed af skýjaþjónustum með **Function as a Service**(Faas) Platform
- Missum mikið af undirliggjandi stjórn en mikið af management overhead er séð um fyrir okkur
- Hafur gott elasticity
- Dæmi
  - AWS Lambda
  - Azure Functions
  - Google Cloud Functions



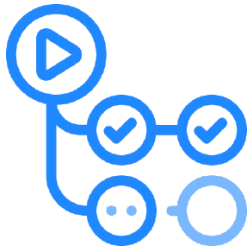
Enabling Technology

Kubernetes

- Kubernetes getur hjálpað okkur með
  - Container Management, DevOps, Rollouts and Rollbacks, Scalin, Infrastructure as Code, Secrets and Configuration, Load balancing, self healing, recovery
- Býður upp á marga kosti en einng mikið overhead
- Hægt að keyra Kubernetes clusters:
  - Locally með Kind
  - On Premise
  - Azure Kubernetes Service (AKS)
  - Amazon Elastic Kubernetes Service (EKS)
  - Google Kubernetes Engine (GKE)

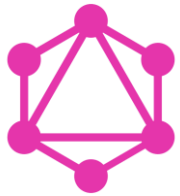
Enabling Technology

So Much More








Microservices Paint Points

- Mikið overhead
  - Bæði tæknilegt og fræðilegt
- Flækjustig
- Debugging erfiðara
- Monitoring erfiðara
- Tracing erfiðara
  - Þurfum að log-a trace-id fyrir hvern lið í workflow
- Testing erfiðara
  - Erfitt að test-a microservices saman
- Service modeling er erfitt
  - Erfitt að ná stærð og domain boundarires service-a rétt
  - Of stórt of við missum kostina við microservices
  - Of lítil og við missum cohesion og independent deployability
- Kostnaður
  - Kostnaður við mörg service
  - Kostnaður við tækni (K8s, Kafka etc.)
- Security
  - Fleir service og tækni == fleiri attack service
  - Mikið af upplýsingum að flæða á milli service-a
- Reporting
  - Getum ekki lengur joinað töflur til að gera skýrslur
  - Streaming (t.d. Kafka) getur hjálpað hér
- Latency
  - Langar synchronous request keðjur geta hægt á kerfinu
- Data Consistency
  - Erfitt að tryggja consistency á milli service-a
  - Distributed transactions eru erfið (og ætti að forðast)

Microservices Paint Points

https://www.youtube.com/watch?v=y8OnoxKotPQ&t=93s&ab_channel=KRAZAM

Premature Decomposition

- Oft er gert þau mistök að fara beint í microservice architecture án þess að skilja domain-ið og context boundary-in nægilega vel

- getur leitt til mikillar tregðu
  - Alltaf að þurfa að breyta, bæta við, fjarlægja, endurskipulegja services

- Oft betra að byrja á monolith
  - Í byrjun eru mikið af breytinum að gerast hratt
  - Domain model-ið er illa þekkt og stanlaust að breytast
  - Erfitt að gera rétt service boundaries
  - Teymi eru oft lítil
  - Betra að byrja á monolith og færa sig í microservices þegar þess þarf.

- Rome wasn‘t built in a day
  - Microservices gerast ekki á einum degi
  - Maður nær architecture-num aldrei rétt við fyrstu tilraun.
  - Við þurfum iteration
  - Oft gott að brjóta kerfi upp í microservices hægt og smátt.

OOP principles eiga líka við um microservices

- SRP
- Encapsulate what varies
- Rule of three
- YAGNI
- KISS
- Data encapsulation
- Program to an interface not implementation
- Favor Composition over Inheritance
- Loose Coupling
- …