SDFI

STABIL OG ROBUST MODULÆR INFRASTRUKTUR

Moderniseret Datafordeler

Dato: 11. September 2023
Version: 1.0
Forfatter: Marcus Quistgaard og René Ravn
Kontakt: mwq@netcompany.com, rra@netcompany.com

netcompany

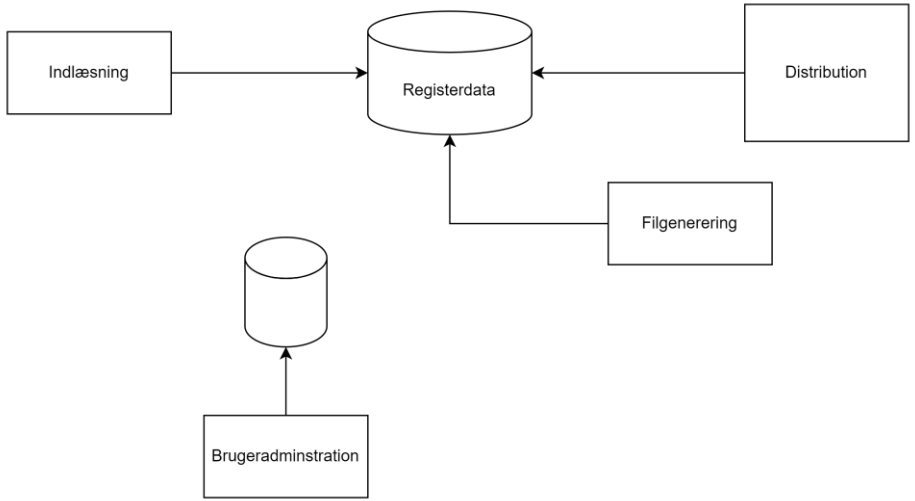
Agenda

- Overordnet system overblik
- Hvad er en container
- Containerorkestrering og overvågning
- Kø-baseret tilgang
- Gevinster og begrænsninger

System overblik

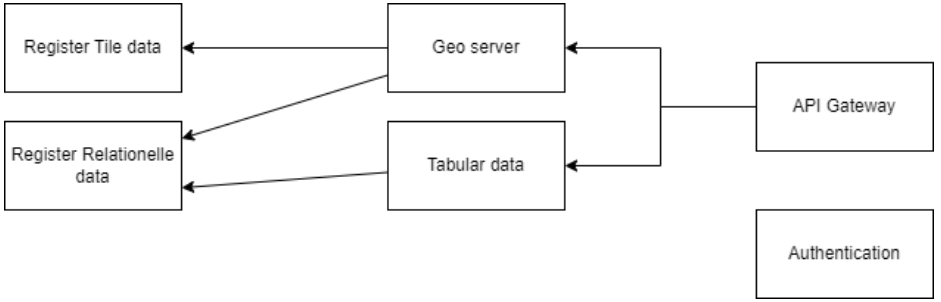
- Indlæsning af registerdata
- Distribution af registerdata via tjenester
- Filgenerering og distribution
- Administration af egne brugerkonti samt af godkendelse af adgange

- Hvert område er selvstændige komponent(er) som hver består af en eller flere container.



Distribution

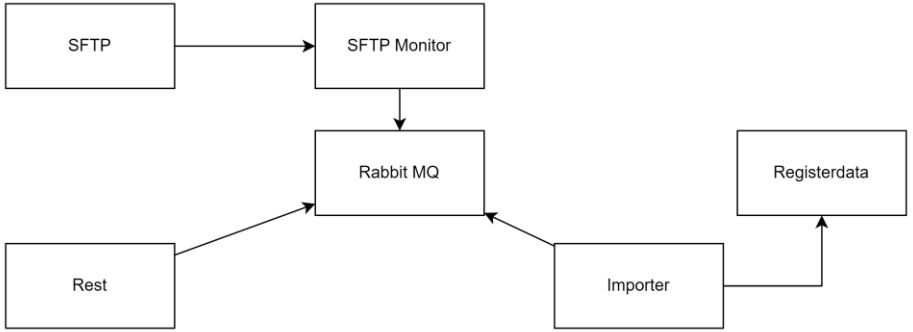
- Gateway
  - Autorisation af brugeren
  - Swagger documentation
  - Metrics/access-log
  - Rate-limiting
  - Mapning af services -> Backend App
- Geo-server
  - Distribution af WMS/WMTS og andre geo formatter.
- Tabular data
  - Distribution af Relationelle data I ikke-geo formatter.
- Authentication -



Modtagelse af data

Stærkt afhængig af indlæsningsflowet

- Standard software til SFTP
- SFTP Monitor behandler de filer som er batch uploaded.
- Rest indlæsning
  - Gemmer indholdet og laver teknisk kvittering. Trigger flow I RabbitMQ
- Importer sikrer konsistent indlæsning af beskeder fra register til Registerdata



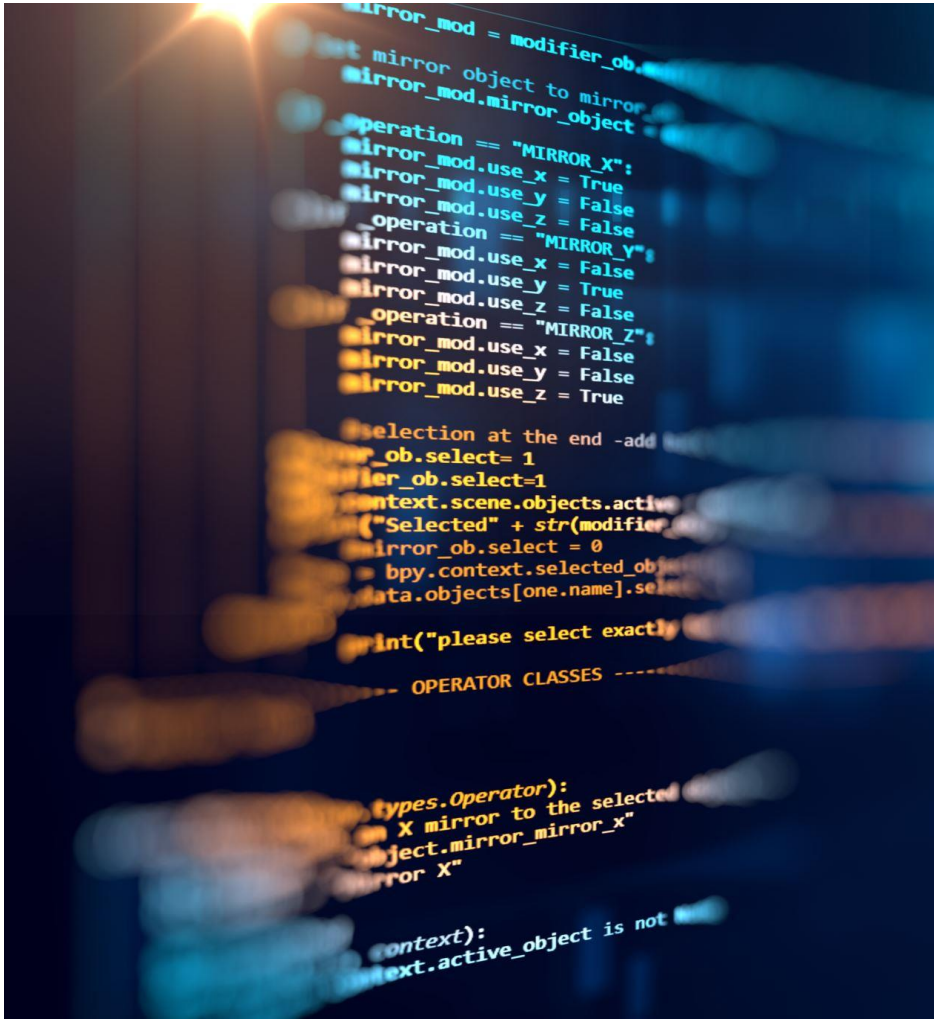
HVAD ER EN CONTAINER

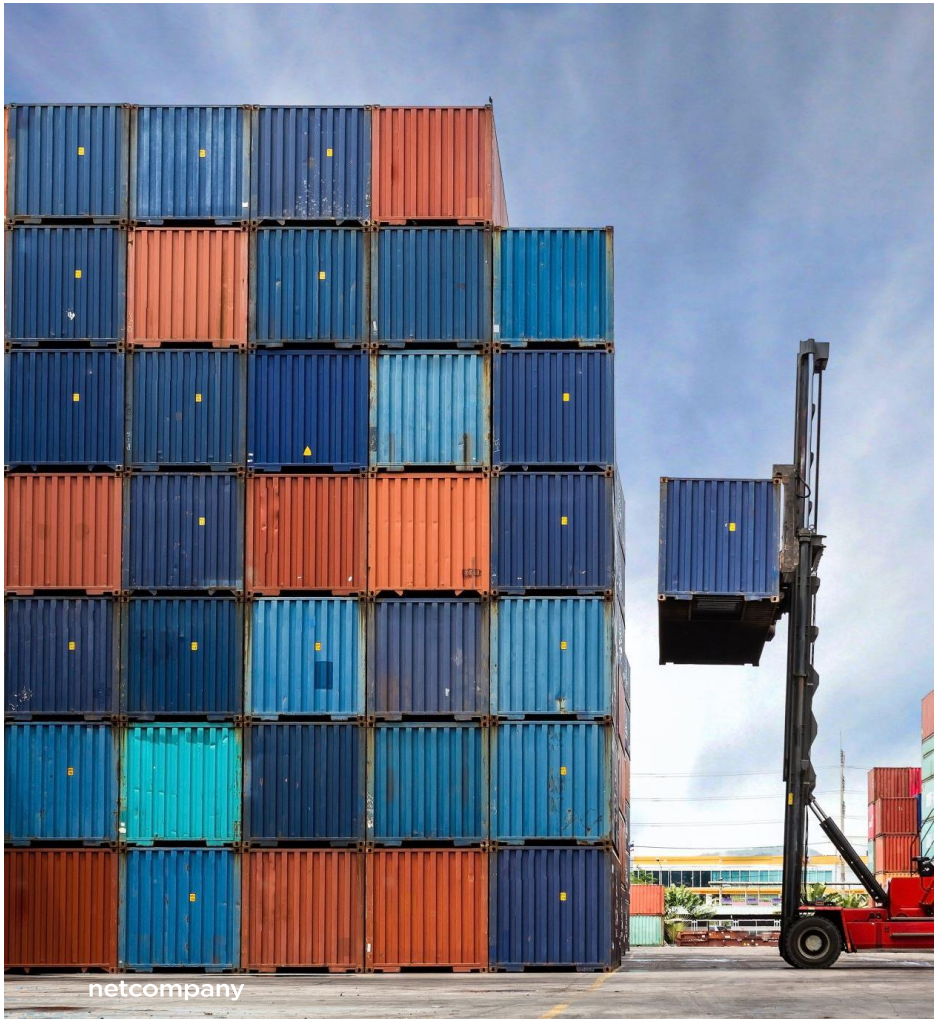
Containering – Hvad er en container

Docker:

“A container is a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another.

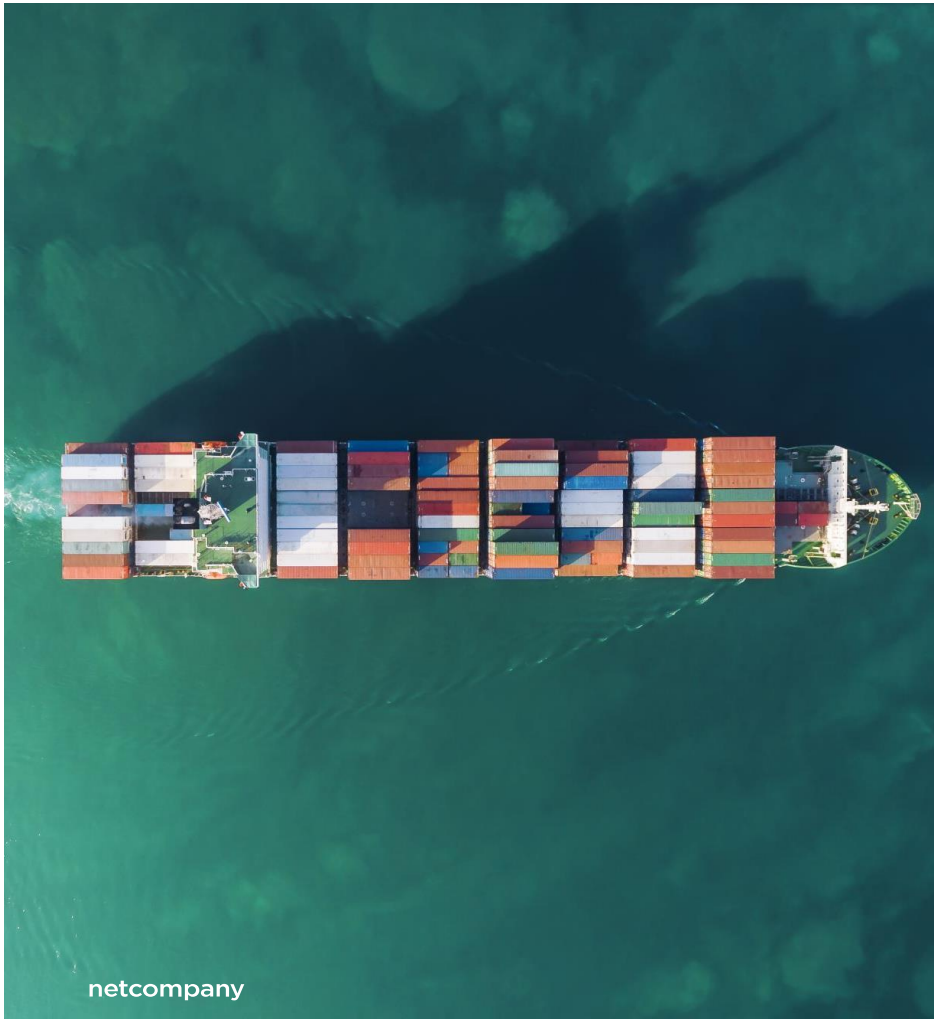
A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.”





Hvad er en rigtig container

- Veldefineret “boks” at indpakke varerne som skal transporteres.
- Ensartede størrelser som afsender kan optimere til pakning, opbevaring og behandling
- Ensartet udstyr til behandling



Containering - Transport

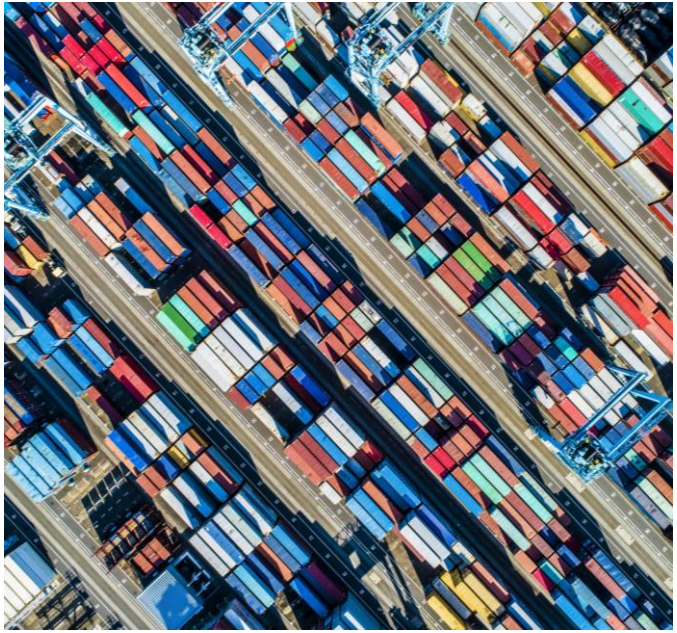
Ensartet størrelse at transportere rundt i verden.

Havne har en struktureret måde at opbevare containeren

Speditøren har ensartet måde at pakke Containeren på transportmidlet hele tiden.

Nem måde at loade hvor den enkelte vare ikke bliver påvirket undervejs

Streamlining i havnen

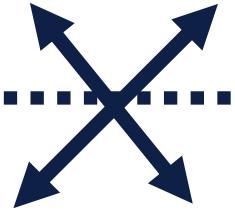


Opsætning og installation

Switchboard

Postgres

Mapserver

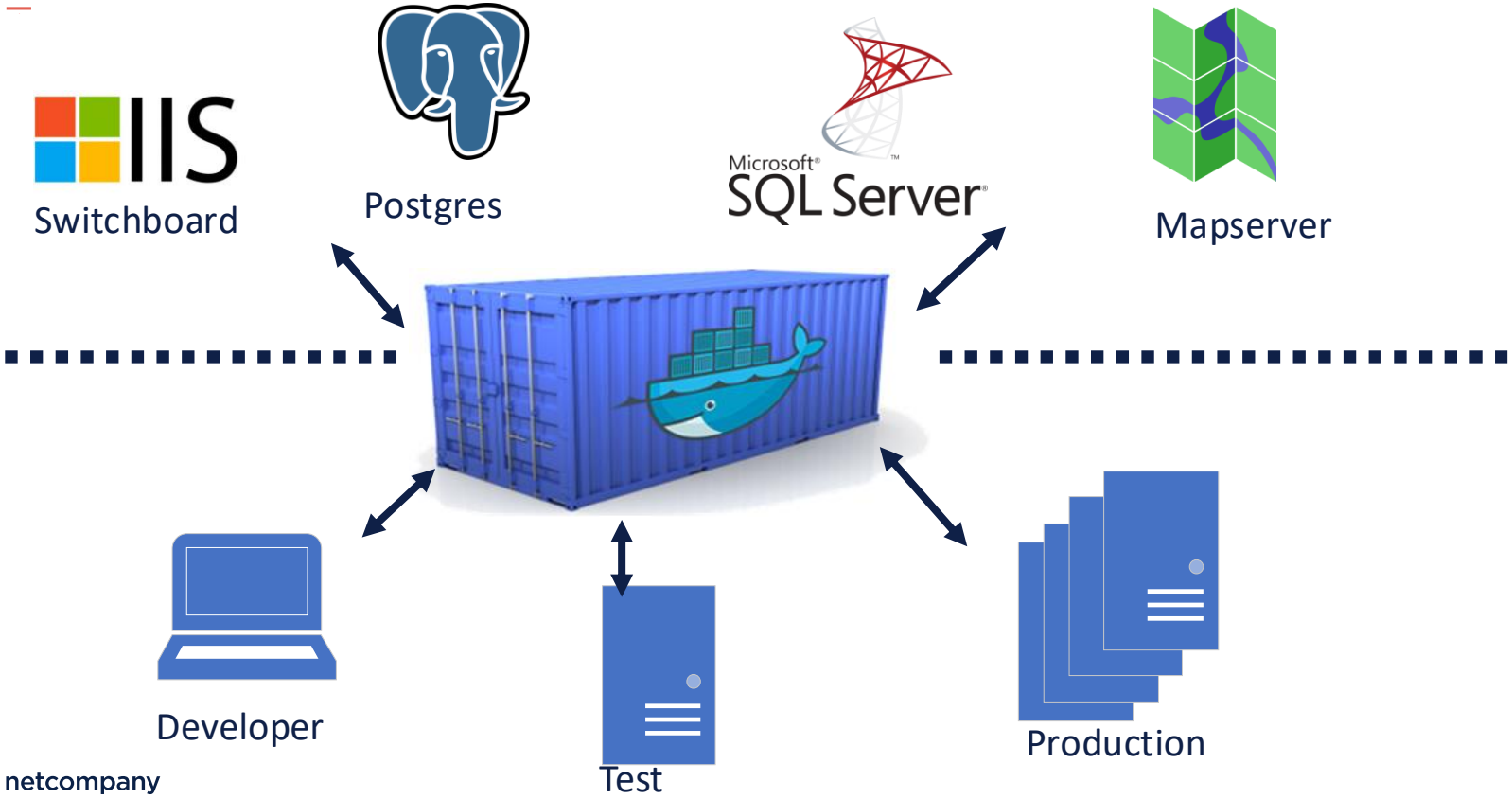


Developer

Test

Production

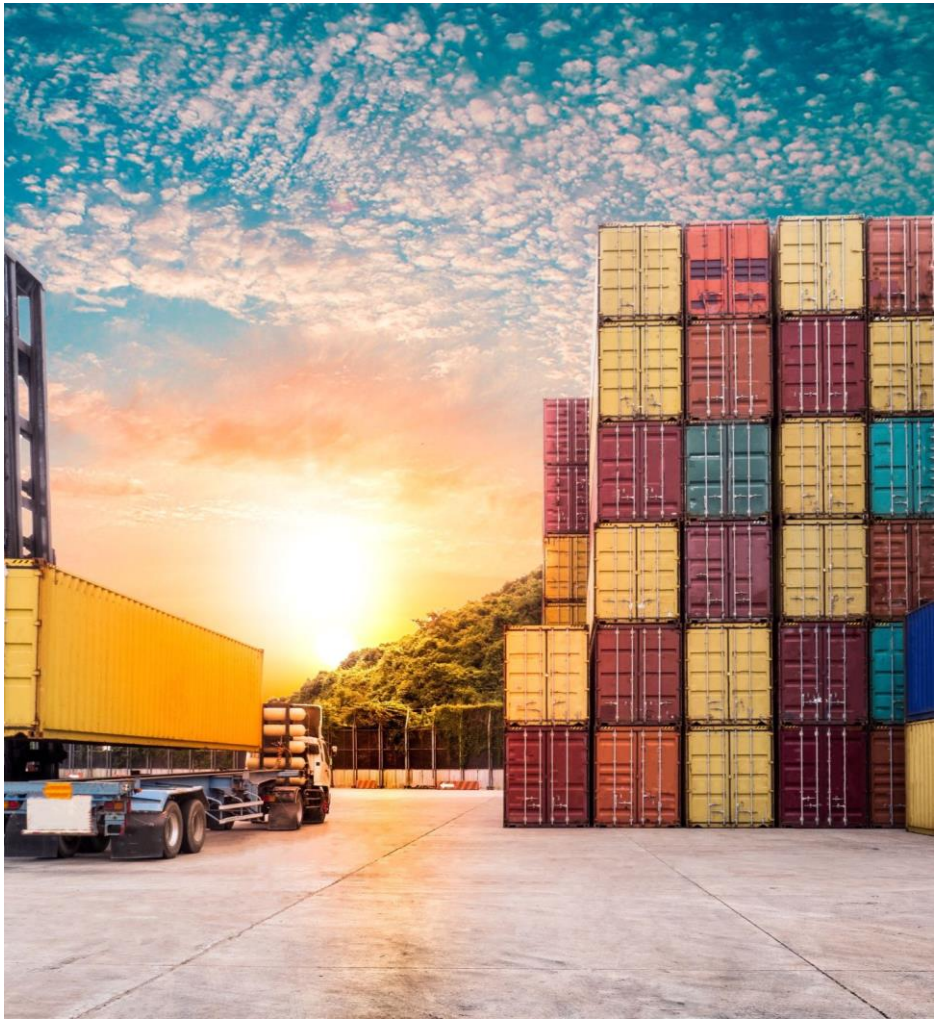
Containerisering af applikationen

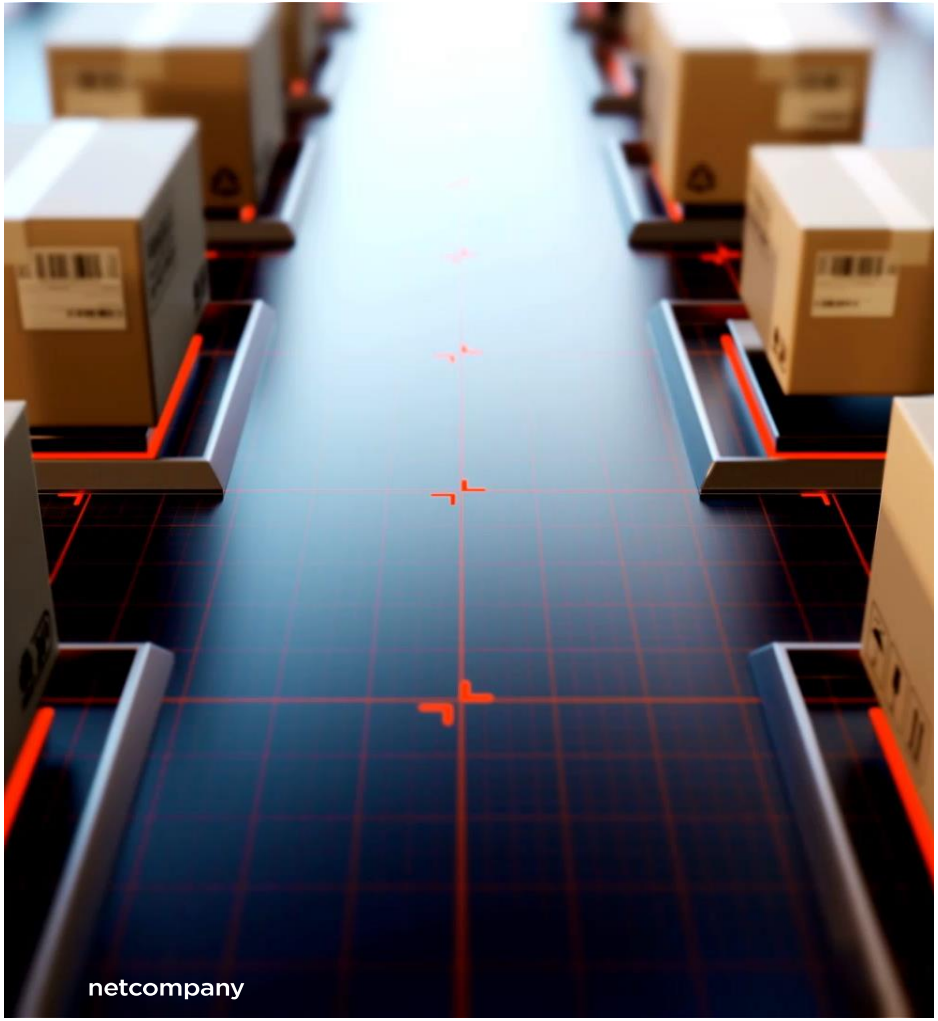


CONTAINERORKESTRERING

Fordele ved container

- Struktureret måde at pakke alle komponenter, uanset programmeringssprog
- Byg container én gang, deploy mange gange
  - Samme byg lokalt som anvendes i prod
- Komponenter kan opdateres og deployes uafhængigt af hinanden





Container orkestrering

- Driftsmæssigt er det en simplere tilgang til de enkelte applikationer.
- Standardiserede metodikker for overvågning af applikationer i containers.
  - Logging, tracing og metrikker
- Nemmere installation og skalering på tværs af servere

Sikkerhedsfordele

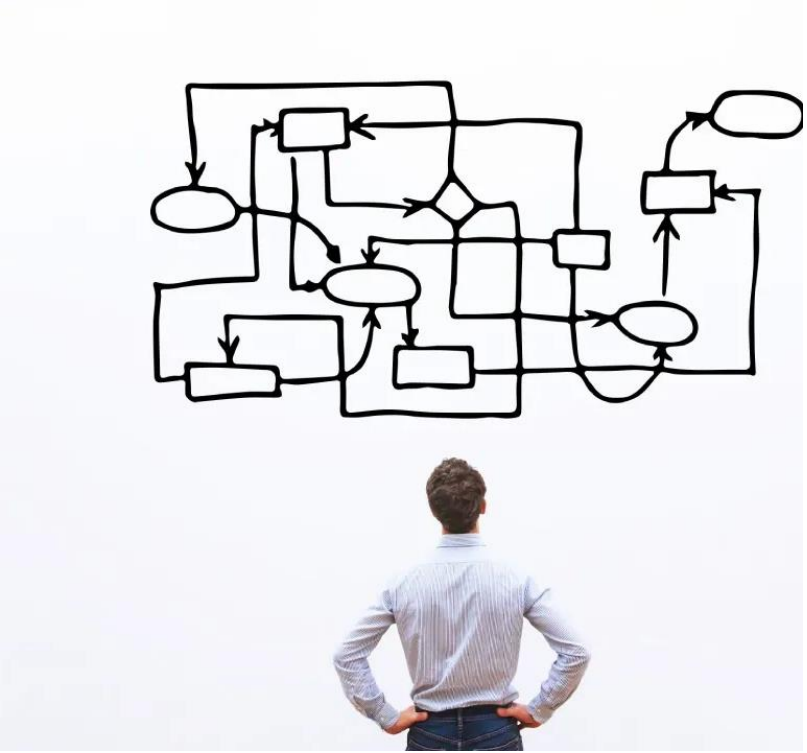
- Containere kan pakkes, analyseres og vedligeholdes ens.
- Automatiseret sikkerhedsscanning af containers.
- Indholdet af en container er begrænset i dens adgang til omverdenen.
  - Den kan ikke modificere indholdet af andre containere eller serveren den kører på.
  - Netværksindstillinger kan specificeres per container.

Procesmæssige gevinster

- Hurtigere lokal udvikling.

  Lettere og hurtigere at få et lokalt udviklingsmiljø sat op.

- Muligt at have helt samme funktionalitet lokalt som på miljøerne.
- Simplificering af release-processen *kan* give kortere byg og releasetid.



Hvad giver containerorkestrering *ikke*

Containerisering giver *ikke* automatisk skalerbarhed. Det kræver at den underliggende infrastruktur også kan skaleres automatisk eller der står allokeret i reserve.

- To eller flere container kan dele samme underliggende servere hvis belastning ligger forskellige tidspunkter
- Nemmere og hurtigere justering og tilpasning af systemet hvor der er behov for skalering.

Container eller ej

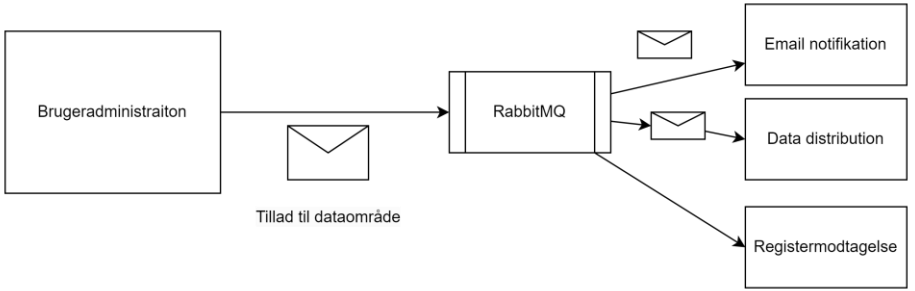
- Målet er at alle *stateless* komponenter vil blive containeriseret.
  - *Stateless* komponenter kan nemt skaleres vertikalt ved at tilføje container af samme type.
  - *Stateful* komponenter skal kunne fungere i cluster

- Container eksisterer både for Windows og Linux, og er i praksis de facto en Linux ting.
  - .Net Core og Microsoft SQL server understøttes på Linux docker.
- Databaser kan blive lagt ind i en container på udviklingsmiljøer, men ikke i miljøer som har eksterne anvendere.
- Det er ikke umiddelbart ønskbart at containerisere samtlige dele af systemet.
  - Microsoft AD / ADFS
  - Windows specifikke applikationer såsom FME kan give problemer

KØBASERET TILGANG

Kø-baseret arkitektur

- En komponent kan lægge en besked på køen og færdiggøre dens arbejde, uden at behøve at bekymre sig om modtager.
- Andre applikationer kan lytte på relevante beskeder og baseret på hændelsen, udføre dens egen logik.
- Ligger basis for hvordan systemet designes.
- Eksempel på flow →



Fordele ved Event

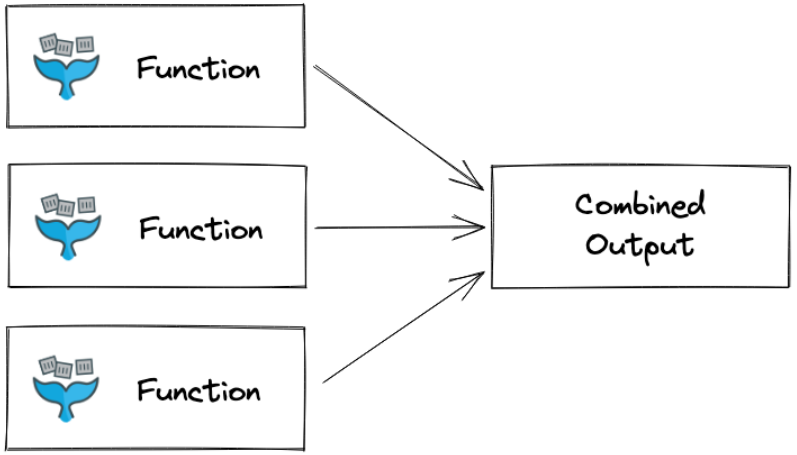
- Undgår punkt-punkt afhængighed mellem de forskellige komponenter
  - Løst koblet moduler som hver især kan driftes
- Kan være mere modstandsdygtig overfor driftsforstyrrelser. Mindre konsekvens at en komponent fejler.
- Opsplitning af ét enkelt flow giver ikke en samlet hurtigere gennemførsel af flowet. Det tillader bedre distribution og fejlhåndtering
  - Tunge opdateringer og beregninger kan eventuelt udskydes til senere.

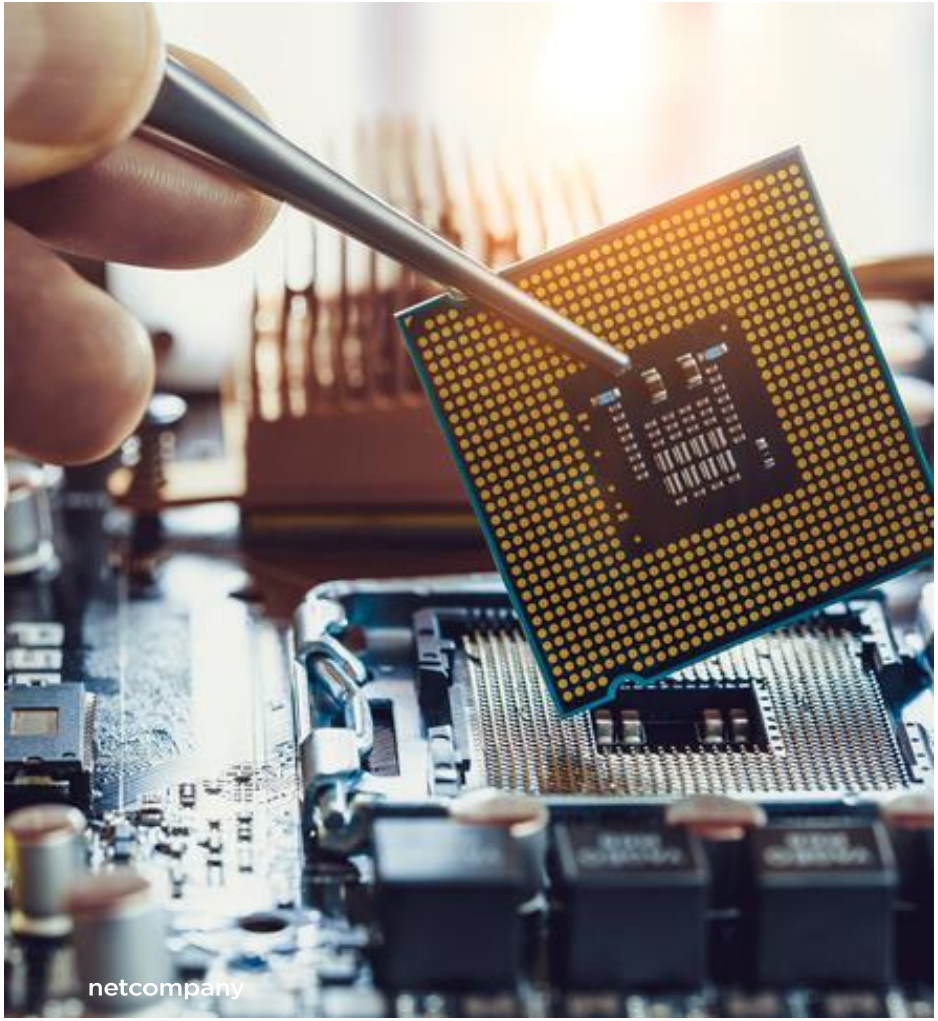
Udfordringer ved beskeder

- Garanteret afsendelse af besked. Flows skal designes med det i minde.
  - Beskeder kan være afsendt, Afsender får ikke kvittering for modtagelse af et eller flere andre dele af systemet
- Beskeder modtages i tilfældig rækkefølge – Designet skal tage højde for det
- Sikre modtagelse af beskeder
- Mere kompliceret sporing af hvad en handling har eller skal afføde i andre dele af løsningen

Skalering af komponenter omkring køen

- Begge ender af køen kan skaleres efter behov
  - *Multiple sources* og *multiple sinks* i en *point-to-point* kø-arkitektur.
- Flere instanser af samme komponent kan dermed håndtere beskeder der ligger i kø samtidig.
  - *Competing consumers* muliggør automatisk balancering af arbejde mellem flere instanser af samme komponent



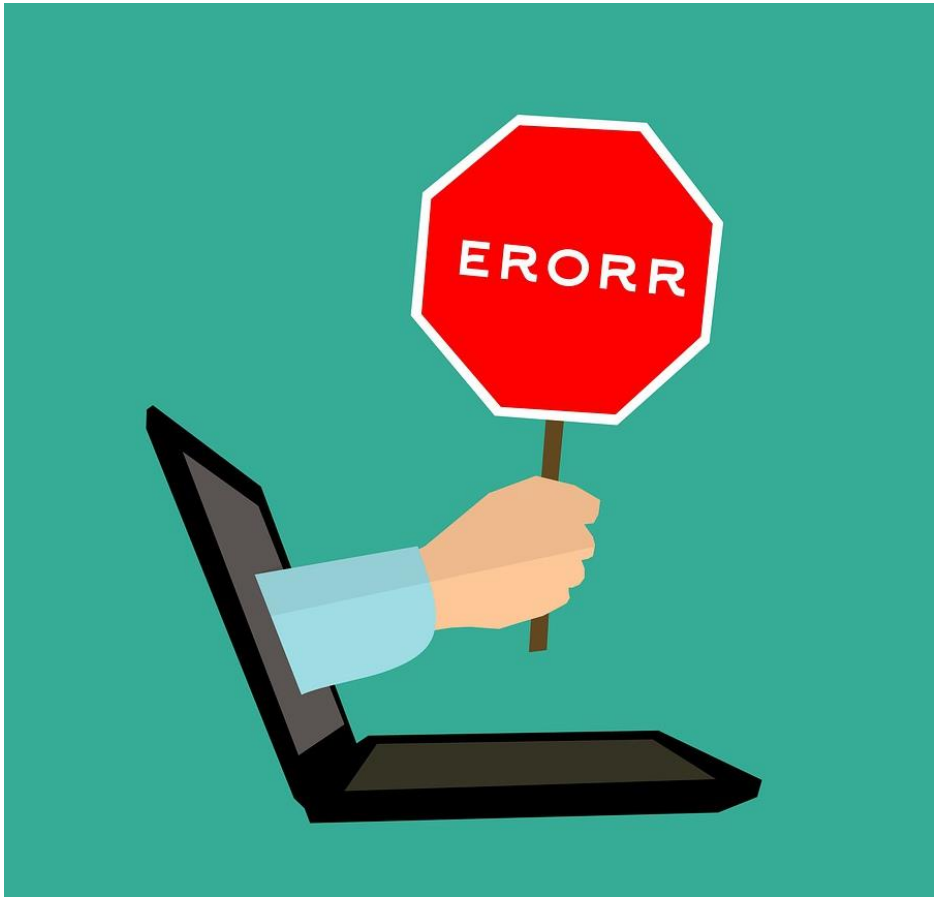


Kø-arkitektur og opdatering af komponenter

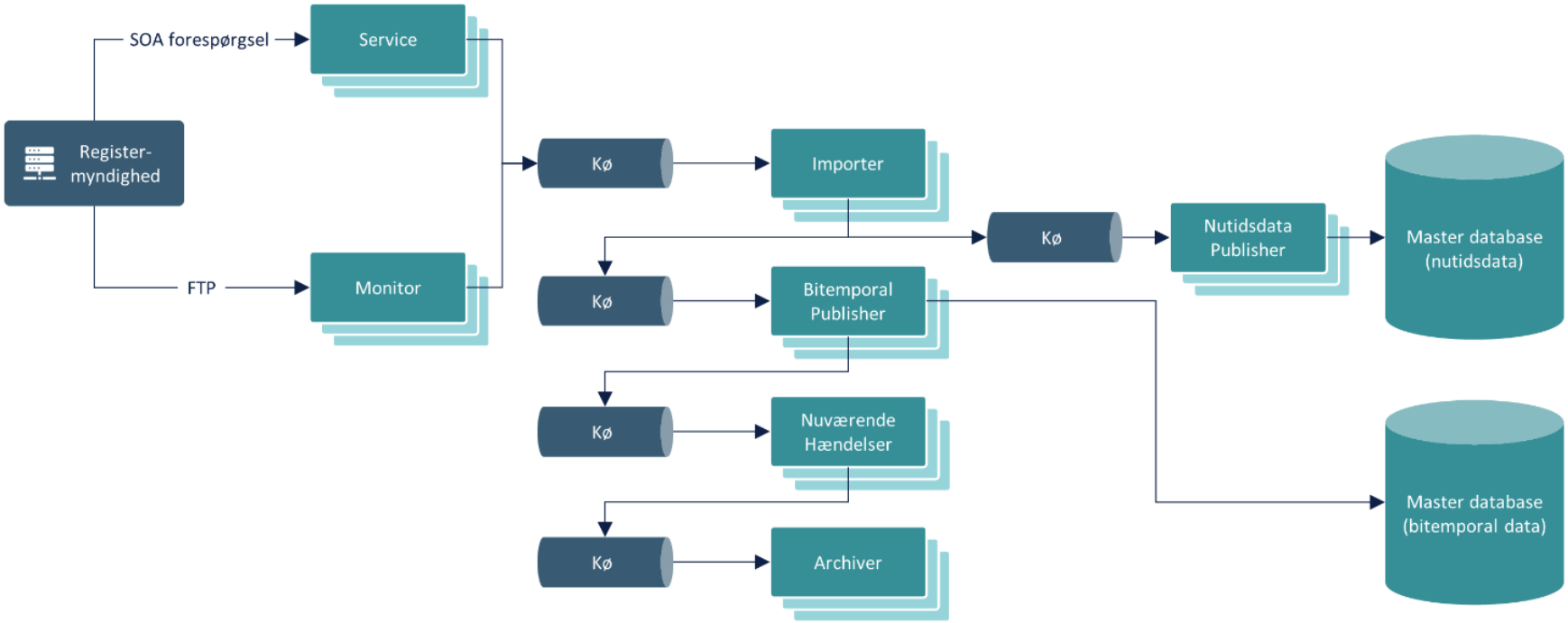
- Komponenter kan opdateres ved at udføre en *opskalering* med opdaterede komponenter og så *nedskalere* hvor de gamle komponenter fjernes.
  - Dette sikrer tilgængelighed under opdateringen. Beskeder vil kontinuerligt blive processeret.
  - Hver komponent kan dermed versioneres og driftes uafhængigt af hinanden.

Fejlhåndtering via køen

- Indbygget mulighed for at genprøve at udføre en given handling.
  - Beskeder i køen kan gensendes eller genafvikles.
  - Er dele af komponenten ikke tilgængelig så kan beskeden blot forblive på køen.
  - Dobbeltmodtagelse af samme Forretningsbesked på grund af asynkron behandling af handling.
- Beskeder kan forblive i køen ved fejl
  - *Exactly-once* vs. *At-least-once* garantier.
- Individuelle dele af løsningen kan dermed fejle uden at hele løsningen er ramt.



Gennemgang af eksempel



Opsummering af gevinster

- Containere er en praktisk måde at pakketere applikationer.
- Containerisering af komponenter tillader let og smidigt at opdatere og skalere komponenter.
- Den underliggende infrastruktur er nødt til at være allokeret på forhånd som muligvis kan bruges lidt mere dynamisk.

- Kø-arkitektur medvirker til separat ansvarsområde for komponenter, på samme måde som vi kender Datafordeleren i dag.
- Event-baserede flows kan give en hurtigere og mere stabil end-to-end processeringstid.


netcompany
www.netcompany.com