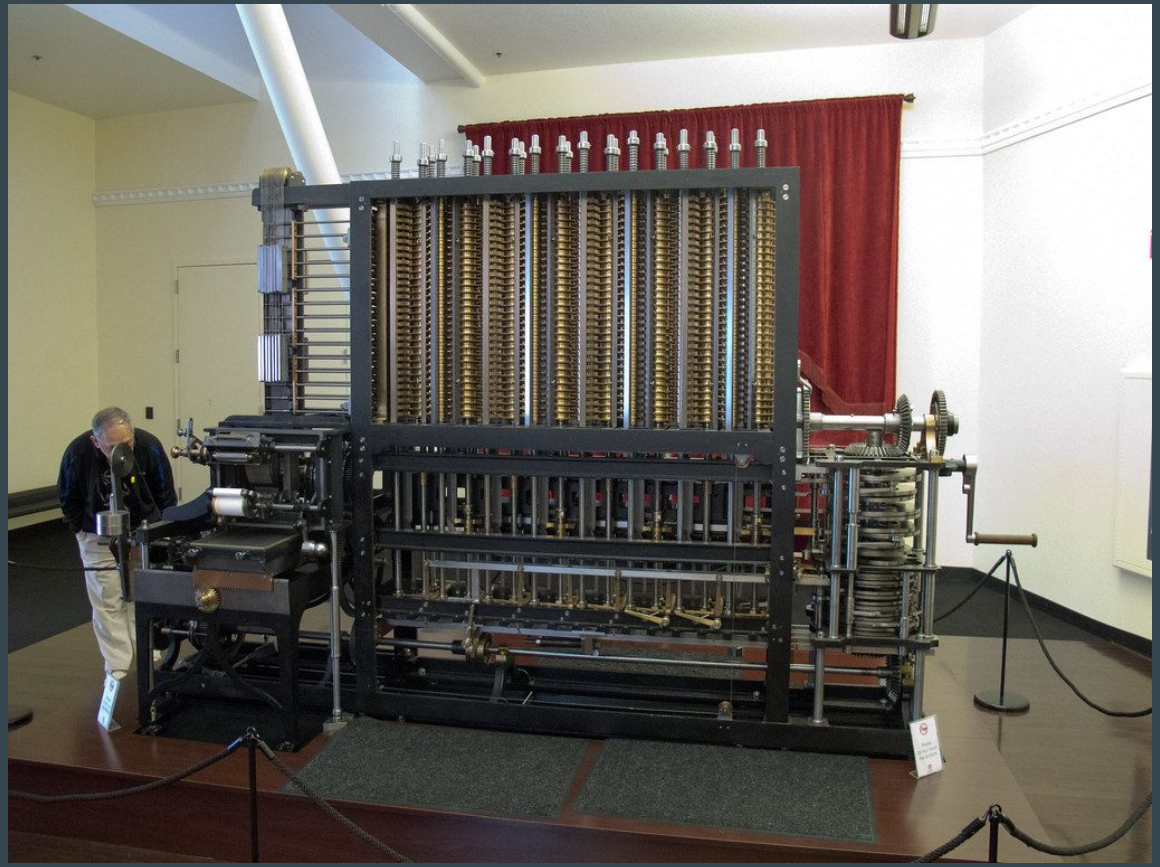
Containerization

Mats Lindh, mats.lindh@gmail.com



Babbage’s Difference Engine No. 2 (1822) (serienummer 2)

Kilde: Eget bilde (CC-BY)

A Commodore 64 has helped run an auto shop for 25 years

BY LEE MATHEWS 09.29.2016 :: 11:14AM EST

162 SHARES

Release date: August 1982

Kilde: geek.com

Dagens plan

- Noe helt annet
- Maskinvare og drift av tjenester
  - Hvor kommer vi fra
  - Virtualisering
  - Cloud hosting
- Containerization
  - Hva
  - Hvorfor
  - Begrensninger
  - Docker

An @AnTheMaker · 1m

just found this in one of my old projects

```
51
52    $array = array(); // array
53
54
```





Welcome to the third post in our series on Python at scale at Instagram! As we mentioned in the first post in the series, Instagram Server is a several-million-line Python monolith, and it moves quickly: hundreds of commits each day, deployed to production every few minutes.

Kilde: Instagram Engineering

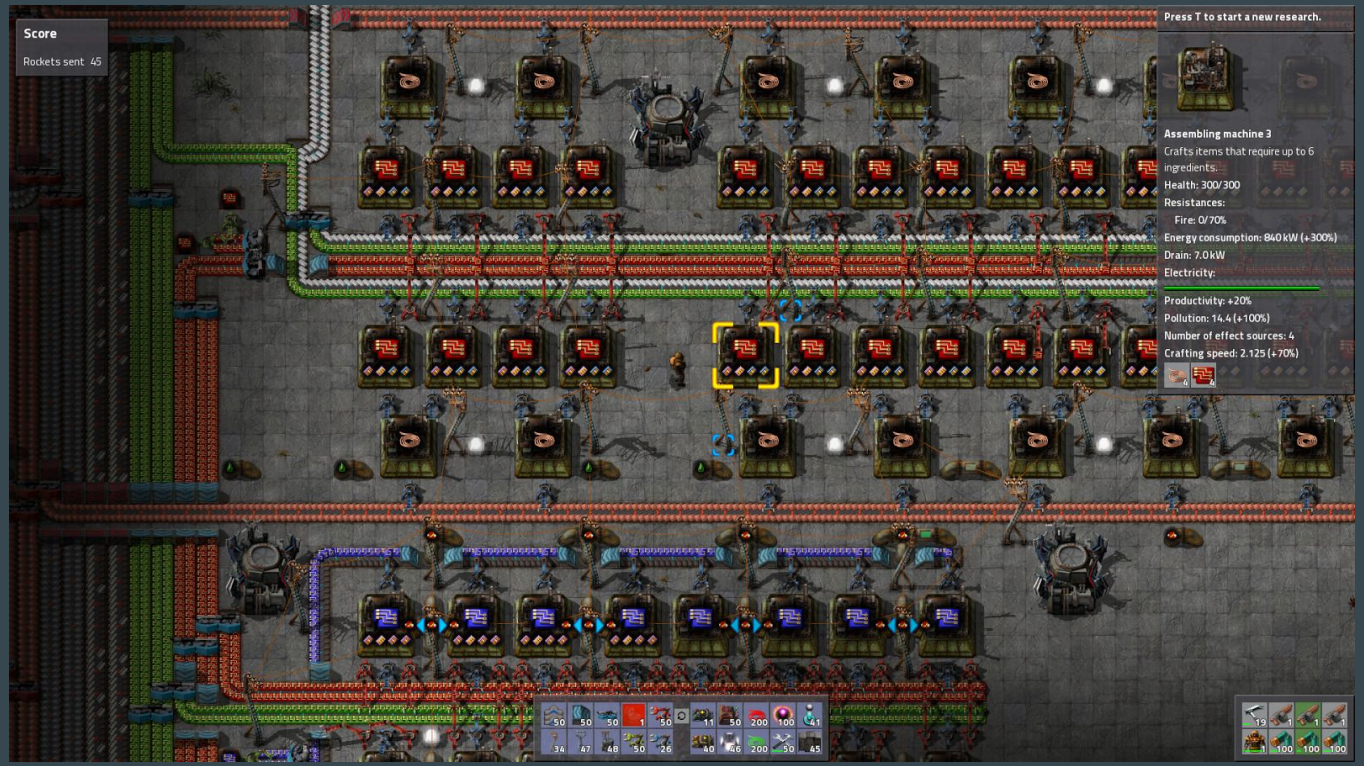
“In my opinion, an extremely valuable skill to learn is when to be **efficiently inefficient**.

There are times where you know you won't be reusing this piece of logic and so hard-coding it in is fine. There are times where you think you won't be reusing a piece of logic and so hard-coding it in may become a problem later on. There are times where you're right, and there are times where you're so, so wrong.

Unfortunately, this is something very, very hard to teach.”

Kilde: Zeromatter / reddit

Testing av spill - Factorio



Integrasjonstesting av spill er notorisk vanskelig - tilfeldigheter og input fra andre spillere påvirker ofte resultatet slik at det er vanskelig å reprodusere, og det kan være snakk om store datamengder som må verifiseres

Factorio - Sterkt forenklet - “Bygg din egen fabrikk og forsvar den mot ulumske fiender”

Testene er i praksis - “last denne tilstanden (savegame), sørg for at interne verdier er satt til ..., gjør følgende, og sjekk at resultatet er ..”

The Automation of Factorio

Kjør alle testene med coverage, slik at dere vet hva dere IKKE har testet!

Video om code coverage

Dagens plan

- Noe helt annet
- Maskinvare og drift av tjenester
  - Hvor kommer vi fra
  - Virtualisering
  - Cloud hosting
- Containerization
  - Hva
  - Hvorfor
  - Begrensninger
  - Docker

Kravstiller - Ruter

Ruter søker etter en erfaren, teknisk orientert ressurs til rollen som kravstiller for utviklingsleveranser innen digitale tjenester for salgs og betaling i Ruter.

Kravstilleren skal støtte produkteieres arbeid med å kartlegge og dokumentere funksjonelle og ikke-funksjonelle behov og krav som vil være grunnlag for utvikling og testing av nye digitale løsninger. I tillegg vil kravstilleren ha en viktig rolle i å bidra til å kartlegge og dokumentere krav til utstyr som skal anskaffes og tas i bruk for nye løsninger, og bidra i planlegging og gjennomføring av anbudskonkurranser for anskaffelse av utstyr til nye digitale løsninger.

Kravstiller - Ruter

Sentrale forutsetninger for å lykkes i denne rollen er som følger:

- Dokumentert minimum 5 års erfaring med prosjektledelse eller kravspesifisering
- God funksjonell og teknisk kompetanse og forståelse av kompleks arkitektur
- Erfaring med offentlige anskaffelser i Norge, inkludert utarbeidelse av kravspesifikasjoner
- Erfaring med *iterative leveranser* i komplekse løsninger
- Skal kunne bruke Jira og Confluence i daglig virke

- Utarbeide/dokumentere detaljert løsningsarkitektur med løsningsskisser i henhold til x arkitekturprinsipper og i tett samarbeid med leverandør, systemeier og forvaltere både i drift og forvaltningsorganisasjonen.
- Utarbeide løsningsarkitektur med løsningsskisser tilpasset x infrastruktur (installasjonsdiagram)
- Bistå ved spesifisering av utviklings-, akseptansetest- og produksjonsmiljø i henhold til utarbeidet løsningsarkitektur.
- Kvalitetssikring av leverandørers løsningsforslag og (tekniske) leveranser
- Utarbeide spesifikke ikke-funksjonelle krav for løsningene og *påse at de er testbare*
- Bidra i kritikalitetsvurdering for applikasjoner
- Utarbeide mindre ROS analyser.

Vi ser etter deg som:

- har erfaring med teknisk testing av Java/Kotlin baserte mikrotjenester, hendelsesorientert arkitektur via Kafka og backends basert på Javascript/Typescript/React.
- har erfaring med brukergrensesnittesting, API-testing, forventningstesting/ kontraktstesting, ytelsestesting og generell akseptansetesting.
- har erfaring med monitorering og feilsøk gjennom moderne observabilitetsverktøy.
- har jobbet med skybaserte plattformer og/eller Kubernetes.
- har minimum 5 års relevant arbeidserfaring. Vi ser etter en senior med faglig tyngde

PERFECT IS THE ENEMY OF GOOD

Dagens plan

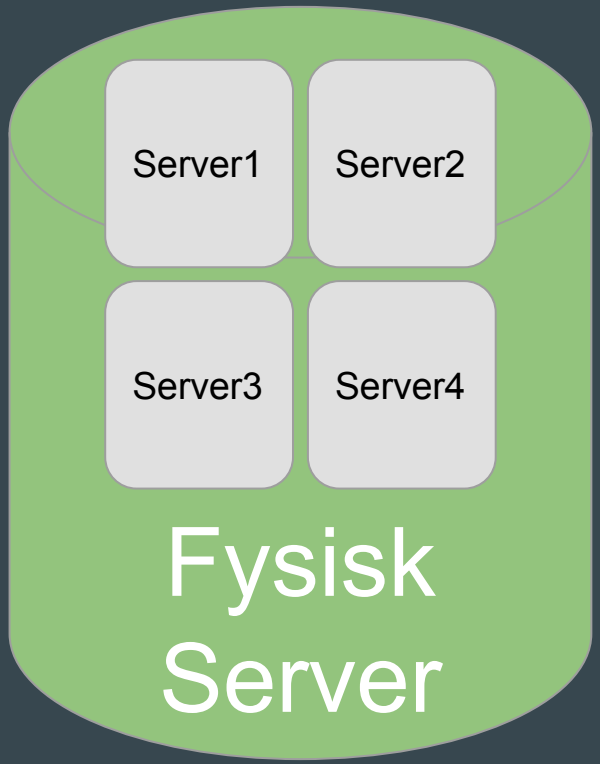
- Noe helt annet
- Maskinvare og drift av tjenester
  - Hvor kommer vi fra
  - Virtualisering
  - Cloud hosting
- Containerization
  - Hva
  - Hvorfor
  - Begrensninger
  - Docker

Dagens reise

Fysiske servere

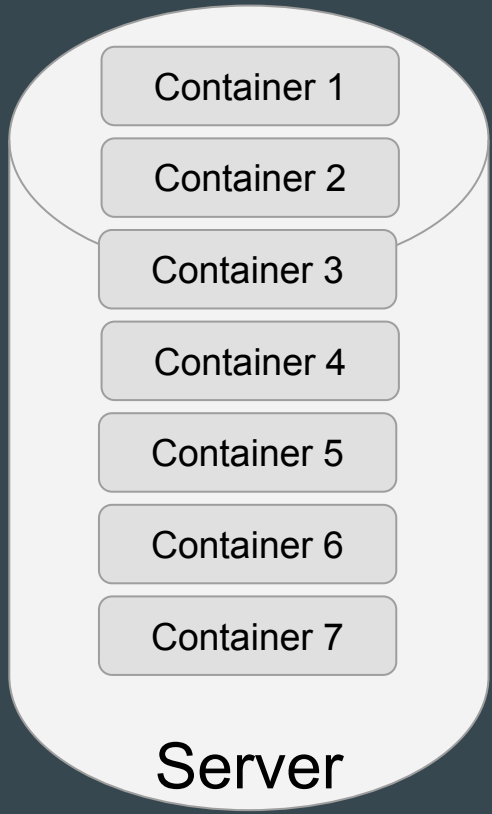
Til godt utpå 2000-tallet

Virtuelle servere



Fra ~2005

Containere



Fra ~2014

Maskinvare og Drift av tjenester

Her var vi

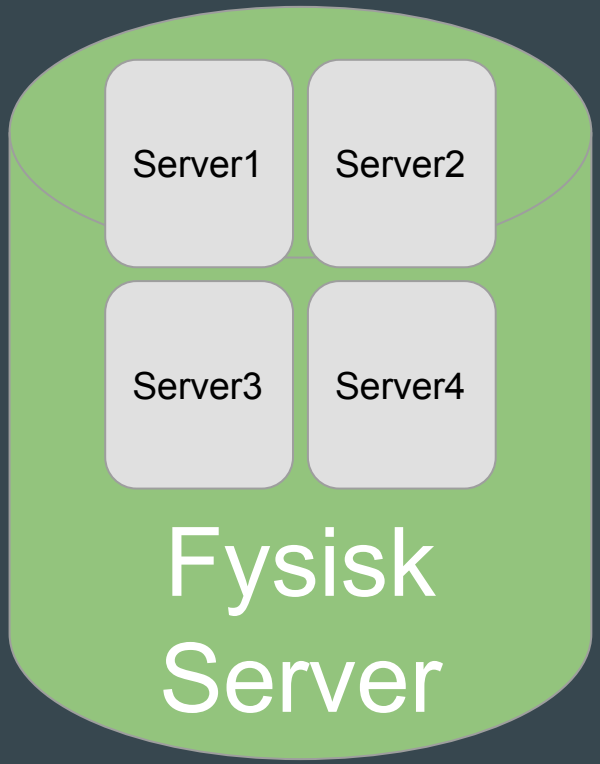
Kilde: CC-SA-BY / torkildr, flickr

Så begynte vi å **virtualisere**

“Serveren” vår var ikke nødvendigvis lenger

en fysisk maskin

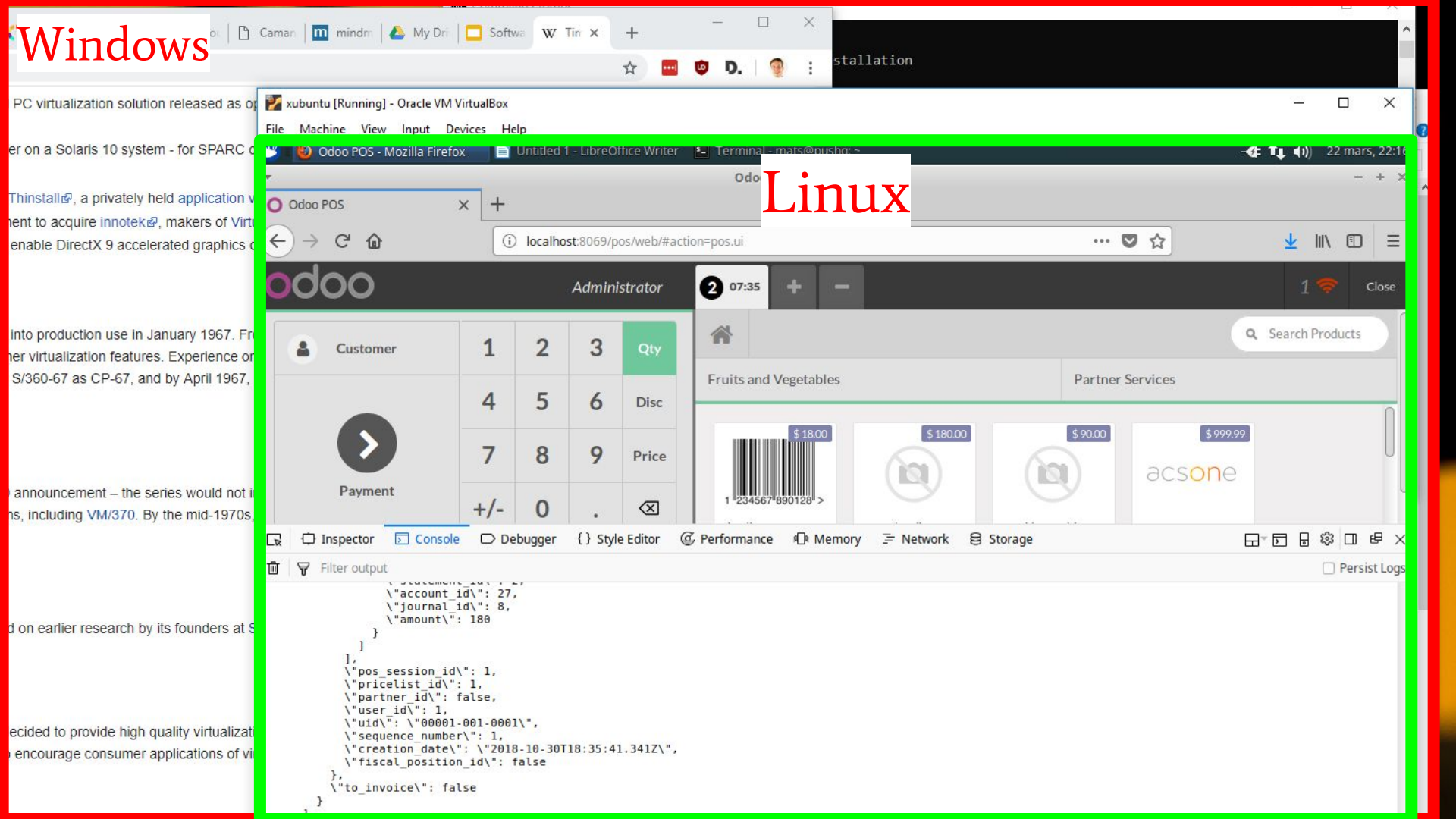
En fysisk maskin kunne plutselig **kjøre mange virtuelle servere** - og holde de helt separat fra hverandre


Server1
Server2
Server3
Server4
Fysisk
Server

En virtuell maskin - en maskin som kjører “inne i” en annen maskin

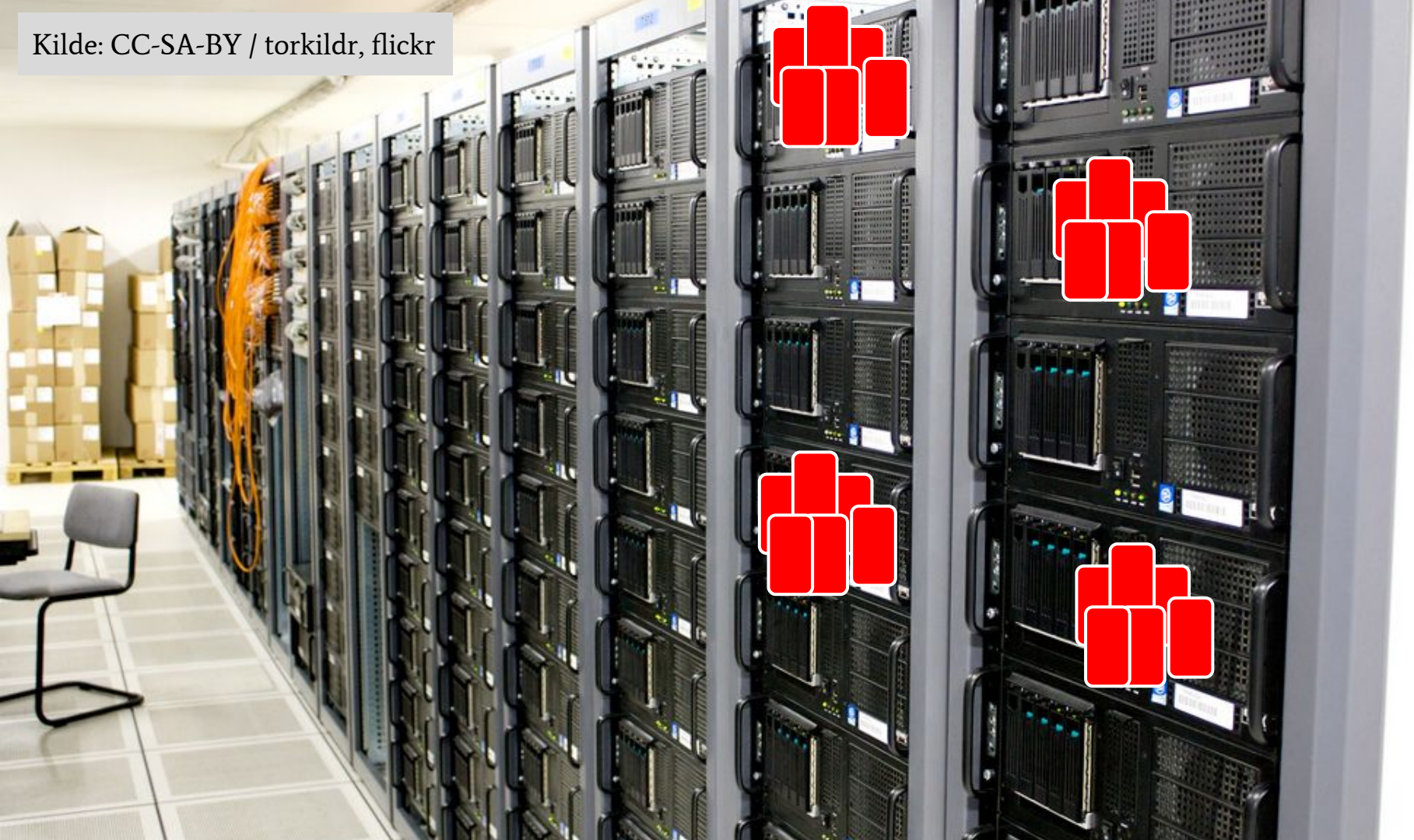
Dette gir oss mer effektiv utnyttelse av maskinvaren vår, og gjør at vi ikke trenger å kjøpe nye servere hver gang vi trenger en ny maskin

Dette kan du teste på din egen maskin - f.eks. med VirtualBox


Windows
xubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Linux
Odoo POS
localhost:8069/pos/web/#action=pos.ui
Administrator
Customer
Payment
Qty
Disc
Price
Fruits and Vegetables
Partner Services
Search Products
Inspector
Console
Debugger
Style Editor
Performance
Memory
Network
Storage
Filter output
Persist Logs

Virtualisering gjør at vi lettere kan tilpasse oss behov i organisasjonen, ettersom vi ikke må skaffe ny fysisk maskinvare hver gang vi skal ha et nytt miljø

Men vi er fortsatt her nede i kjelleren


Kilde: CC-SA-BY / torkildr, flickr

“Cloud Hosting”

There is no cloud

It’s just someone else’s computer

Andre eier maskinvaren som de virtuelle maskinene våre (eller tjenestene våre) kjører på - døden for det lokale serverrommet

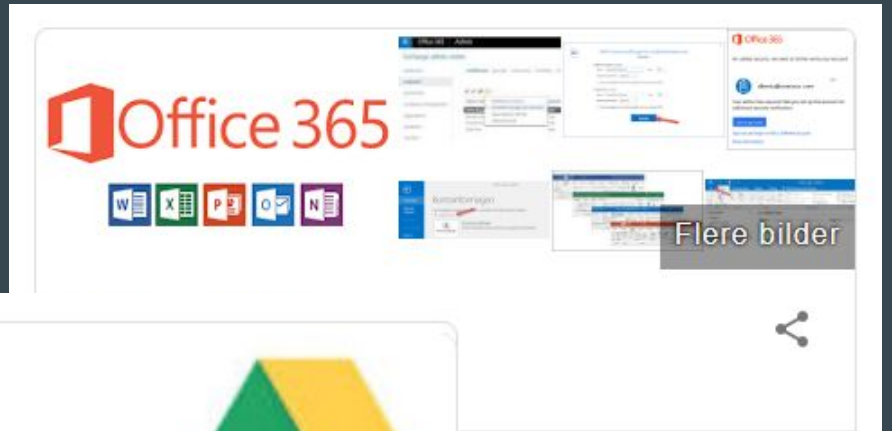
I 2010

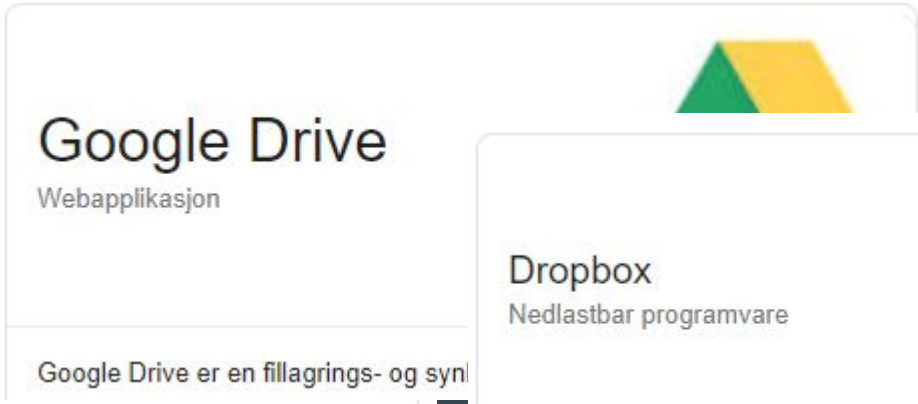
Mail i en typisk bedrift? Stort sett Microsoft Exchange på en server stående på et rom hos bedriften

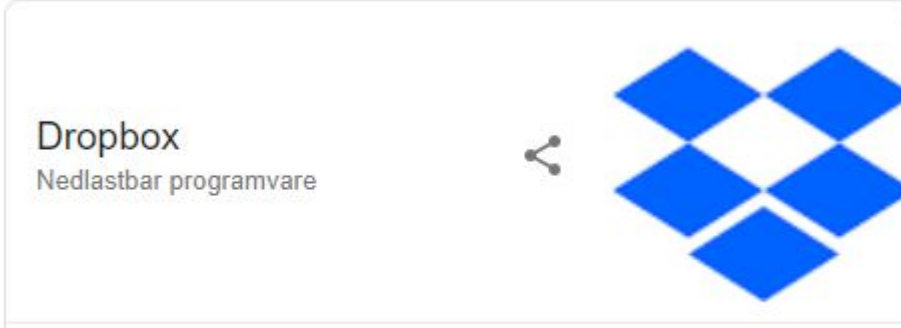
Filer? En windows-server på samme rom med en del diskplass

NÅ


Flere bilder
Gmail
Gmail er en gratis e-post-, POP3- og IM
først utviklet av Paul Buchheit. I Storbrit
det offisielle navnet Google Mail. Gmail
april 2004, og man måtte da bli invitert a
kunne opprette en konto. Wikipedia
Skapt av: Paul Buchheit
Lanseringsdato: 1. april 2004


Office 365
Flere bilder


Google Drive
Webapplikasjon
Google Drive er en fillagrings- og synl


Dropbox
Nedlastbar programvare

Epost lever ikke lenger “i bedriften” - men leveres fullstendig av en ekstern tilbyder (“i clouden”) - du har ikke noe forhold til den fysiske maskinen som tjenesten din kjører på lenger

Det samme gjelder for maskinvaren for applikasjonene våre - de kjører ikke lenger på en maskin vi eier i et serverrom et sted

I stedet er det mange større aktører som leier ut ressurser etter behov, slik at vi kan kjøpe oss plass hos andre når vi trenger det

Vi trenger ikke lenger bruke tid og penger på drift av fysisk maskinvare, serversentere, etc., og kan tilpasse oss endringer mye raskere enn tidligere (slipper å vente på levering av .., oppsett av .., stresstesting av..)

(og når noe ryker er det ikke vår jobb å fikse det)

Jeg har i praksis postet på Facebook tre ganger

Mats Lindh er her: **Boston Public Library** med **Anette V. Heiberg**.

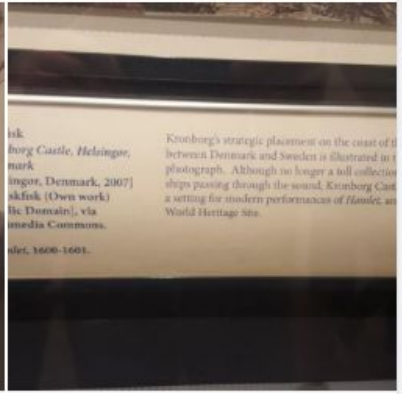
26. oktober 2016 · Boston, USA ·

Tidenes merkeligste sammentreff - jeg og Anette tuslet rundt i Boston for en drøy ukes tid siden, og ramlet innom Boston Public Library, USAs tredje største offentlige bibliotek. Masse fint å se, men når man først er på et sted som har et eget kartrom (som Gunnar Misund har gjort at jeg alltid stikker innom om jeg snubler over noe), så er det jo på sin plass å avlegge et besøk.

Helt til jeg begynte å faktisk lese teksten under det ene bildet og forstår hva som står der, og oppdager at det er mitt eget bilde - fra Wikipedia - som nå er utstilt på Boston Public Library. I en utstilling om kart fra Shakespeare sin tid (og her om Danmark på grunn av Hamlet).

Så om du tar deg bryet med å laste opp bildene dine på Wikipedia og kaste dem ut i Public Domain (som i praksis betyr at de kan brukes til hva som helst, hvor som helst), så kan du plutselig ende opp med å oppdage ditt eget bilde. På utstilling. I Boston.

Her illustrert ved bildet "Overrasket norsk mann fant eget bilde i fremmed land".



Mats Lindh is with Anette V. Heiberg.

July 12, 2017 · Skjærhalden ·

På søndag startet en ny epoke i livet - vi ønsket Leo Vikerhaug Lindh velkommen til verden! Alt står bra til med både mor og barn (og faren har det ikke så aller verst selv må det sies), og nøkkelopplysningene er 3502 gram og 51cm.

En enorm takk til hele personalet på barsel og føde på sykehuset på Kalnes: jordmødre, barnepleiere, barneleger, de som vasket og de som serverte mat; dere har gjort denne opplevelsen til noe helt spesielt og hjulpet oss videre på hvert eneste lille steg som skal tas. Det har aldri vært et negativt svar eller noe som har vært et problem, og vi har aldri følt oss tryggere på noe som er en stor livsendring. Dere vet selv hvem dere er, og dere er alt for mange til at jeg klarer å huske navnet på alle vi har hatt noe med å gjøre .. særlig når man plutselig har blitt foreldre. 🙂 Tusen, tusen takk!

Mats Lindh

February 20, 2018 ·

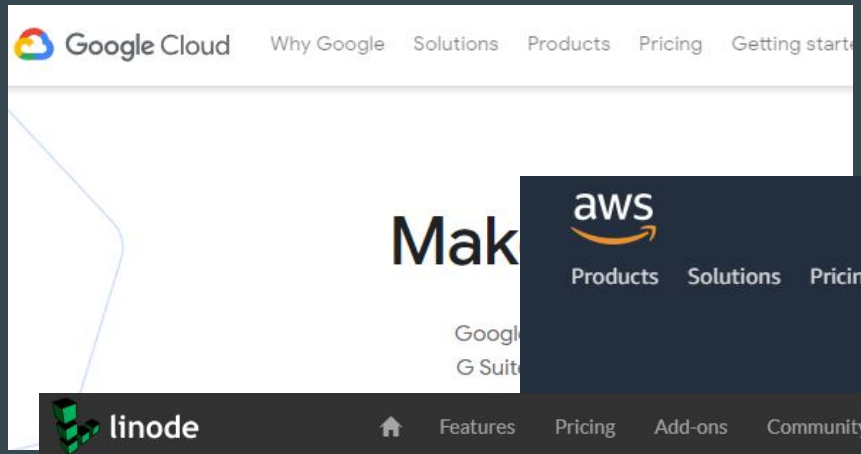
Internett! Jeg trenger et Dell PowerEdge 1800 Power Supply! Enten type redundant (DPS-650BB, FD732, GJ315, KD045, P2591), eller type ikke-redundant (C4797, GD323, TJ785, U2406). Dell sitt beste tips var Ebay, men skulle gjerne hatt det hakket raskere enn det! Ping meg på fornavn.etternavn@gmail.com om det er noe som slenger et passende sted!

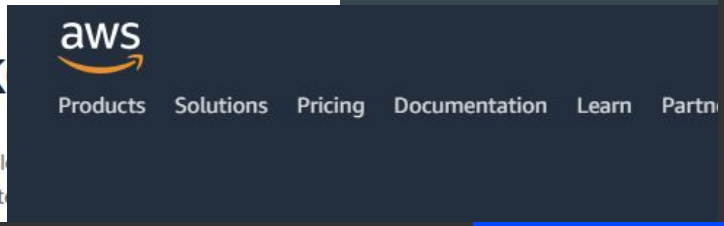
14 Comments 8 Shares

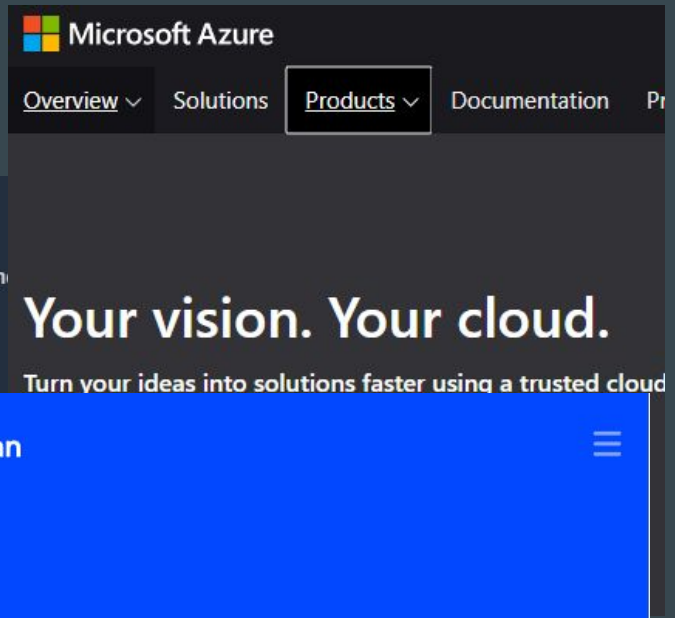
(serveren levde i halvannet år til med et powersupply vi kjøpte på Expert og rewiret - litt det samme som skjedde med Expert som ble rewiret til Power)

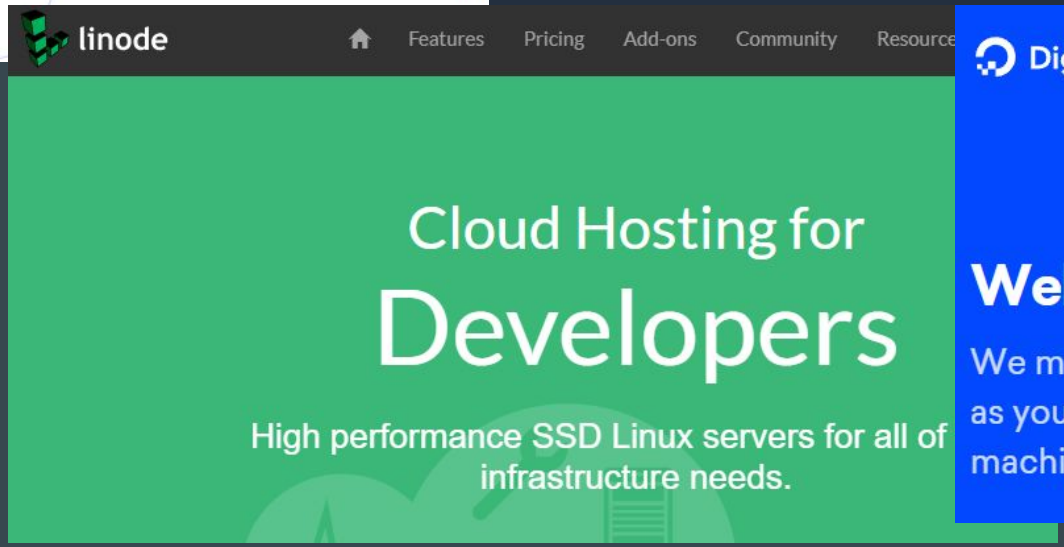
(kidden lever fortsatt, uten at vi har rewiret ham)

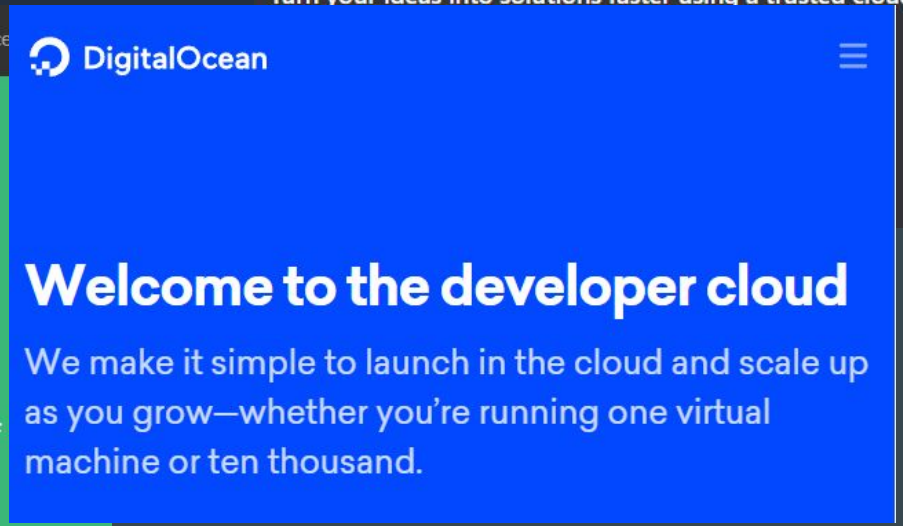
Nå lar vi heller noen andre ta seg av jobben *og risikoen* med de fysiske serverene


Google Cloud
Why Google
Solutions
Products
Pricing
Mak


aws
Products
Solutions
Pricing
Documentation
Learn


Microsoft Azure
Overview
Solutions
Products
Documentation
Your vision. Your cloud.
Turn your ideas into solutions faster using a trusted cloud


linode
Features
Pricing
Add-ons
Community
Cloud Hosting for
Developers
High performance SSD Linux servers for all of
infrastructure needs.


DigitalOcean
Welcome to the developer cloud
We make it simple to launch in the cloud and scale up
as you grow—whether you're running one virtual
machine or ten thousand.

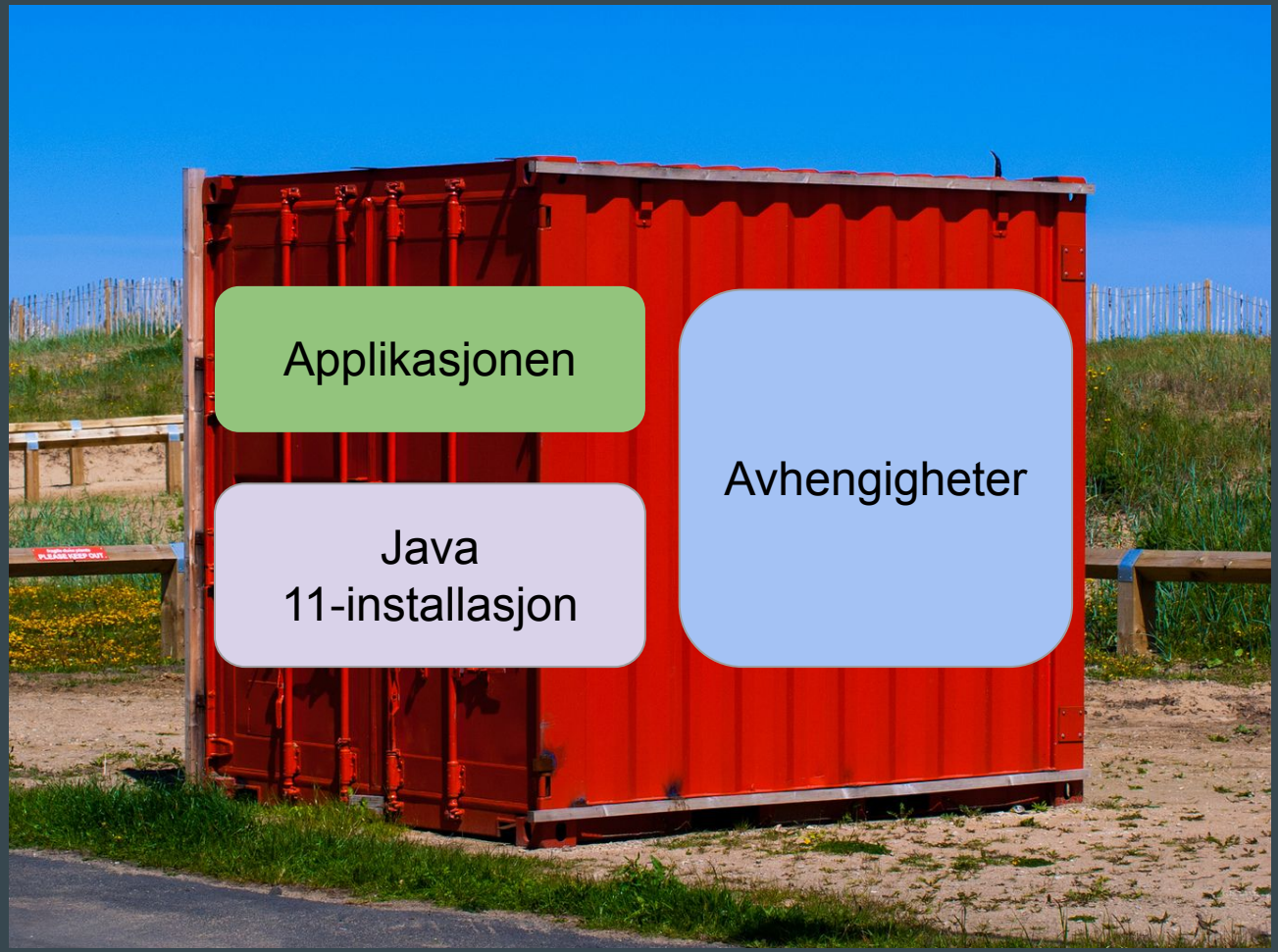
Dette gir LAV investeringskostnad for å komme i gang med et prosjekt - og du kan bygge ut etter hvert som behovet dukker opp!

Dagens plan

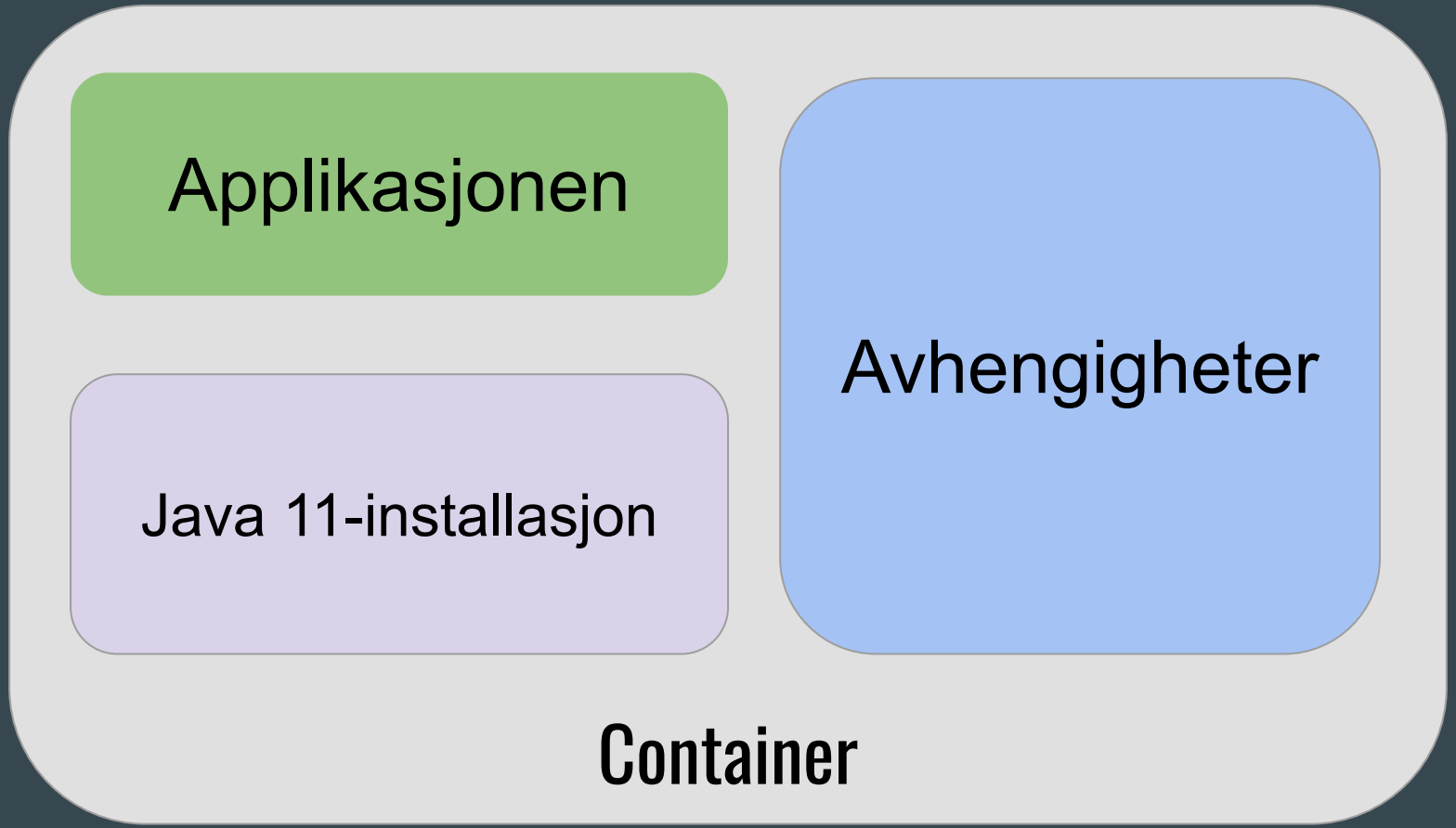
- Noe helt annet
- Maskinvare og drift av tjenester
  - Hvor kommer vi fra
  - Virtualisering
  - Cloud hosting
- Containerization
  - Hva
  - Hvorfor
  - Begrensninger
  - Docker


PLEASE KEEP OUT

Neste steg - i stedet for å få tilgang til en virtuell maskin som vi setter opp og installerer programvaren vår på, bygger vi en egen, liten versjon av “maskinen” vår og sier “kjør denne!”


Applikasjonen
Java
11-installasjon
Avhengigheter

Vi stapper alt som applikasjonen trenger inn i samme boks



Hvordan “maskinen” skal se ut beskrives i et eget format og med en applikasjon som lager “maskinen” vår slik vi ber om.

Hele prosessen kan plutselig automatiseres - “*start med denne maskinen*”, “*gjør disse stegene*” og “*her er ‘maskinen’ slik du ville ha den*”

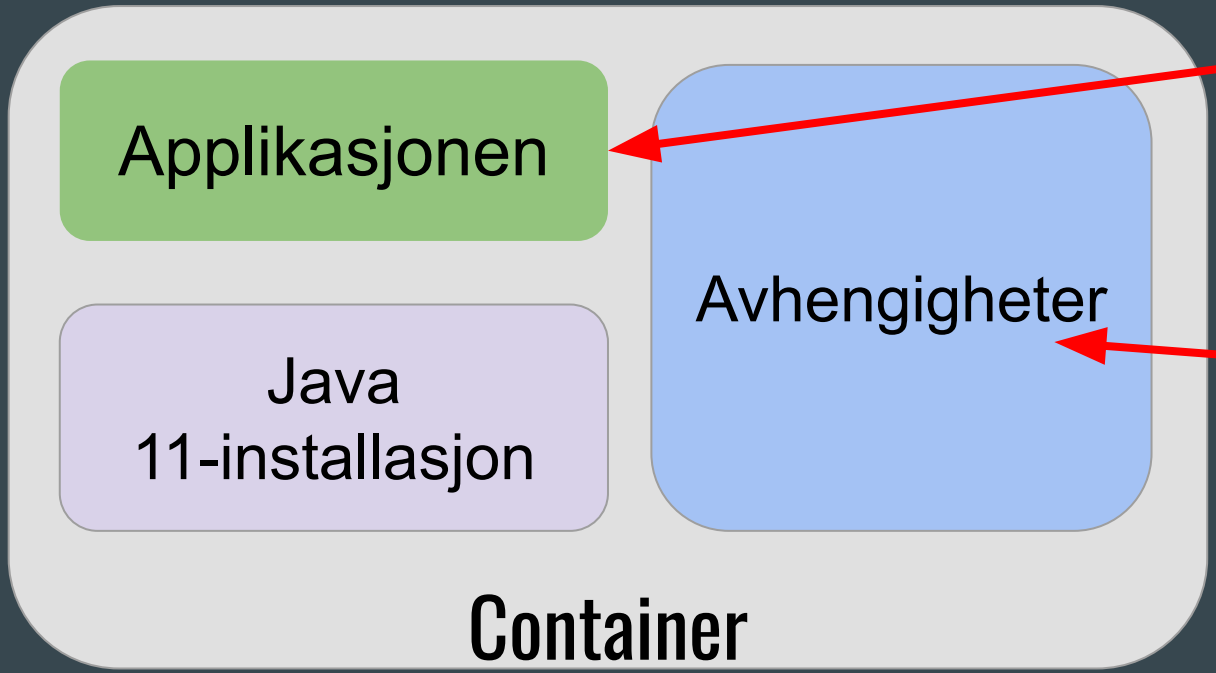

Standardmaskin med ferdig miljø (f.eks. for Java, PHP, Python, etc.)
Kopier inn applikasjonen vår
Installér avhengigheter
Angi hva som skal kjøres ved oppstart
Applikasjon som ferdig container
Disse ligger fritt tilgjengelig på nett på Docker Hub

Resultatet er en

Container

- en lettvekts, “virtuell” maskin

Containeren inneholder applikasjonen, plattformen og avhengighetene - en komplett versjon av alt som må til for å starte opp applikasjonen



Kopieres inn eller kompileres i containeren

Installeres i containeren ved å bruke metoden til systemet for avhengigheter i containeren - f.eks. ved å kjøre gradle, maven, npm, composer, etc.

Containere er ikke fullstendige virtuelle maskiner (teknisk sett) - men for all praktisk bruk kan dere tenke på dem som det

Før:

- Opprett en virtuell maskin
- Logg på maskinen
- Installér nødvendig programvare
- Kopier inn applikasjonen
- Installér avhengigheter
- Sett opp maskinen slik at applikasjonen starter av seg selv ved omstart
- Skriv et dokument som viser hva du har gjort og hvordan noen kan gjøre dette senere
- Glemmer å oppdatere dokumentet når infrastrukturen endrer seg...

Nå:

- Skriv en *programmatisk beskrivelse* som sier hvordan maskinen skal settes sammen
- Be om at "maskinen" din blir bygd etter oppskriften
  - Kopiér inn disse filene..
  - Kjør disse kommandoene ..
  - Kjør denne kommandoen for å starte applikasjonen
- Beskrivelsen er alltid det riktige bildet av hvordan miljøet (og "maskinen" til applikasjonen faktisk er satt sammen

Docker er den mest populære applikasjonen for å håndtere denne prosessen for oss - både for å kjøre og for å bygge "maskinene"

“Byggeinstruksjonene” er en liten fil (“Dockerfile”) som sier hvordan maskinen vår skal settes sammen

```
# base image (python)
FROM alpine:3.12.0

# Kopièr inn filene fra lokal katalog til /app
COPY appenvaar /app

# Tjenesten vår kjører på port 4242
PORT 4242

# Kjør kommandoen “startappen”
CMD [“startappen”]
```

Offentlig maskin som kan brukes som utgangspunkt

Virkeligheten er ofte litt mer kompleks

```
FROM python:3.11-slim as base

ENV PYTHONFAULTHANDLER=1 \
      PYTHONHASHSEED=random \
      PYTHONUNBUFFERED=1

RUN apt-get update && apt-get upgrade -y

WORKDIR /app

FROM base as builder

# Required for building greenlet
RUN apt-get install -y gcc

ENV PIP_DEFAULT_TIMEOUT=100 \
      PIP_DISABLE_PIP_VERSION_CHECK=1 \
      PIP_NO_CACHE_DIR=1 \
      POETRY_VERSION=1.4.0

RUN pip install "poetry==$POETRY_VERSION" greenlet

COPY pyproject.toml poetry.lock README.md ./
COPY appname ./appname

RUN poetry config virtualenvs.in-project true && \
      poetry install --only=main --no-root && \
      poetry build

FROM base as final

EXPOSE 8000

COPY --from=builder /app/.venv ./.venv
COPY --from=builder /app/dist .
COPY docker-entrypoint.sh .
COPY alembic.ini .
COPY alembic ./alembic

RUN ./.venv/bin/pip install *.whl
ENTRYPOINT ["./docker-entrypoint.sh"]
CMD ["gunicorn", "godtiming_main.app:app", "--bind",
"0.0.0.0:8000", "--workers", "4", "--worker-class",
"uvicorn.workers.UvicornWorker"]
```

Vi gjør noe tilsvarende når vi jobber med CI/CD: Eksempel med Github Actions (.github/workflows/foo.yaml)

```
name: install-and-test-xyz-main-api
on: [push]
jobs:
  build-xyz:
    name: "Run tests"
    runs-on: ubuntu-latest
    steps:
```

Containerene er lettvekt (< 500MB) - og kan nå flyttes rundt etter eget ønske - alt som er relevant for applikasjonen (avhengigheter, program for å kjøre applikasjonen, etc.) er

pakket inn i containeren

Containeren har ikke tilgang til filer utenfor sitt eget miljø og er tilnærmet isolert fra omverdenen

Containere er derfor uten varig tilstand (“stateless”) - de kan tas ned, drepes, restartes, etc. - all persistent lagring skjer vanligvis utenfor containeren i form av diskområder som “mappes” opp.

Dette er viktig når vi kommer til hvordan vi styrer containerene våre og sprer de utover flere maskiner!

Lettvekts”maskiner” - hver “maskin” kan ha sitt eget ansvar - vanligvis *en enkelt applikasjon* - uten å ha andre tjenester inne i samme container

Trenger du en databaseserver? Start en databasecontainer i tillegg

En ny applikasjon? Lag en container og start den også på samme server

Så hvordan ser infrastrukturen vår ut nå om dagen?

Bilde: Wikipedia / Hummelhummel

Hvilke fordeler får vi av at vi nå har små containere som inneholder de enkelte delene av det som tidligere var “serveren vår”?

Isolasjon

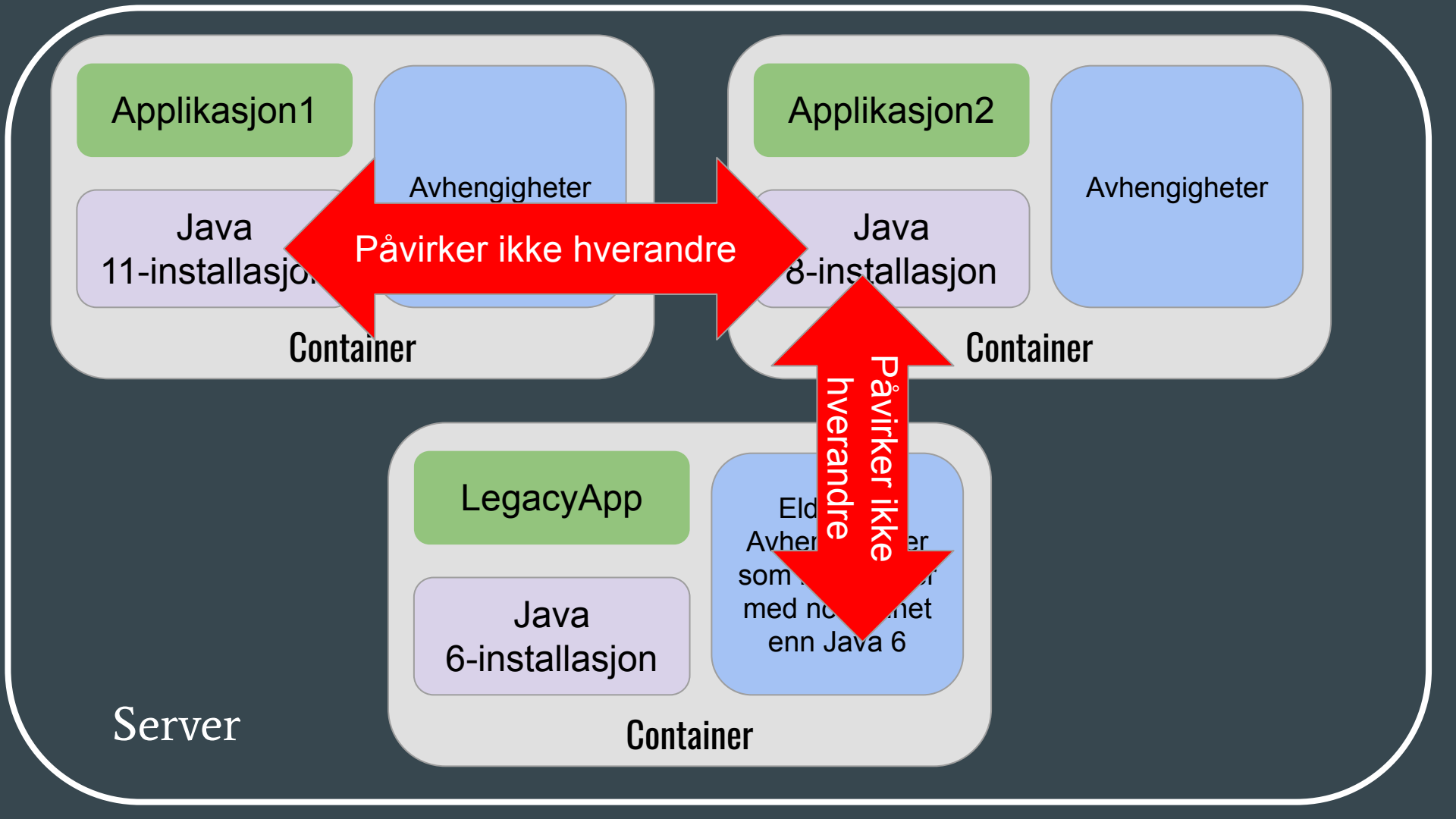
Isolasjon - applikasjonen vår lever for seg selv - ingen andre vet om eller får endre på filer/innhold/etc. som er en del av containeren og applikasjonen vår

SuperApplikasjon vet ikke om MegaApplikasjon - selv om de kjører på samme fysiske maskin - så lenge de kjører i ulike containere.

De kan fortsatte prate sammen via nettet, men de lever i sitt eget miljø - isolert fra andre containere!

Avhengigheter og miljøet i hver container er utelukkende tilgjengelig for applikasjonen i containeren - ingen andre containere benytter seg av eller vet hva som finnes inne i andre containere.

Dette er samme resultat som om hver applikasjon hadde hatt sin helt egne server, bare for seg selv


Applikasjon1
Avhengigheter
Java
11-installasjon
Container
Påvirker ikke hverandre
Applikasjon2
Avhengigheter
Java
8-installasjon
Container
Påvirker ikke hverandre
LegacyApp
Java
6-installasjon
Container
Server

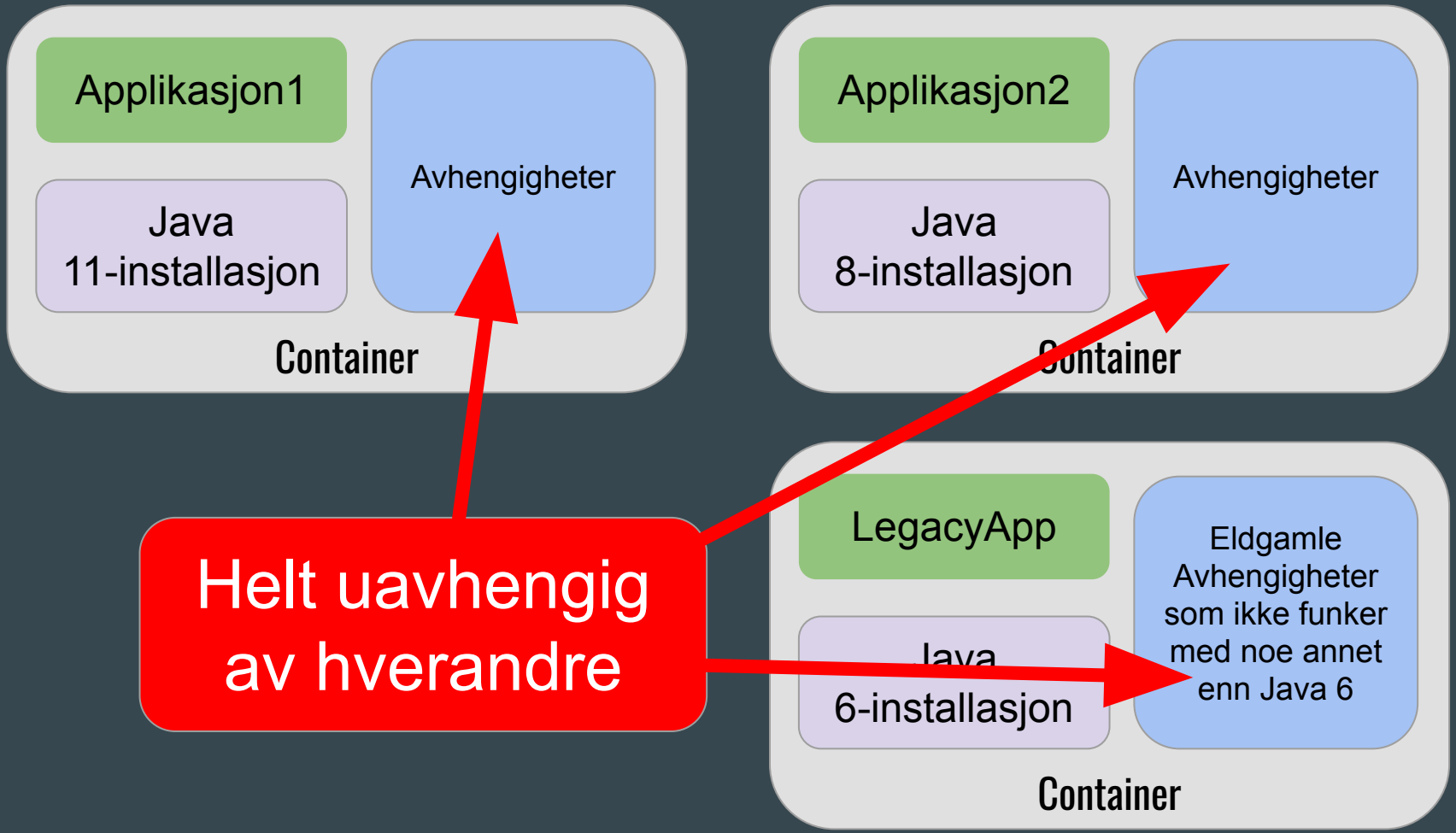
Stabilitet

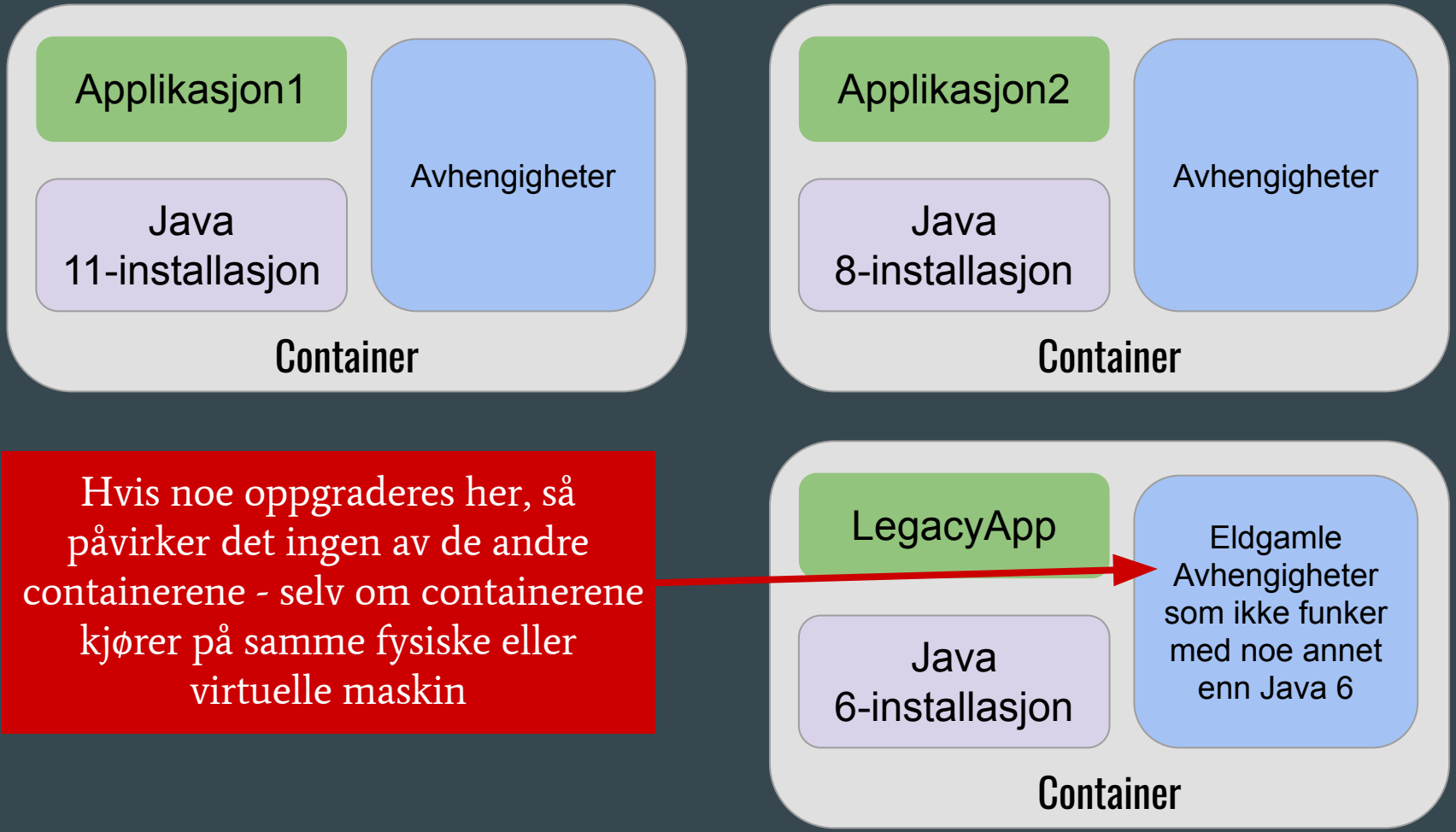
Stabilitet - ingen oppgraderer andre deler av serveren vår - noe som *kanskje*, *tilfeldigvis*, *muligens* påvirker applikasjonen vår

Containeren vår er statisk og kan ikke endres utenfra (men vi kan laste opp en ny versjon og så starte den i stedet om vi vil endre noe).

Endringer i en annen container *påvirker ikke innholdet i vår egen*, og applikasjonen kjører i "sin egen verden" som ikke påvirkes av hva andre gjør

Ved problemer kan *containeren vår restartes med samme innhold* som den hadde da vi startet den for første gang - som å reboote en maskin uten at noen endringer har skjedd på den fra slik den var da vi satte den opp første gang


Applikasjon1
Java
11-installasjon
Avhengigheter
Container
Applikasjon2
Java
8-installasjon
Avhengigheter
Container
Helt uavhengig
av hverandre
LegacyApp
Java
6-installasjon
Eldgamle
Avhengigheter
som ikke funker
med noe annet
enn Java 6
Container


Applikasjon1
Java
11-installasjon
Avhengigheter
Container
Applikasjon2
Java
8-installasjon
Avhengigheter
Container
Hvis noe oppgraderes her, så påvirker det ingen av de andre containerene - selv om containerene kjører på samme fysiske eller virtuelle maskin
LegacyApp
Java
6-installasjon
Eldgamle Avhengigheter som ikke funker med noe annet enn Java 6
Container

Isolasjon gir stabilitet

(som er en tagline som funker fint for kode også)

Reproduserbarhet

“Det virker på min maskin”

Din “maskin” er nå identisk med alle sin “maskin” - det er samme container og innhold som kjører på din maskin, i test eller i produksjon

identisk “maskin” - identisk oppførsel

(men ikke nødvendigvis identisk konfigurasjon (prod vs test database, etc.), så det er fortsatt noen fallgruver)

Containeren vi bygger lokalt er den samme (identisk) som containeren som kjører i produksjon

Min maskin

Der all magien skjer - ting er aldri stabilt og endrer seg fortløpende

Testing

Ustabil programvare som er tiltenkt fortløpende testing og automatisk bygd

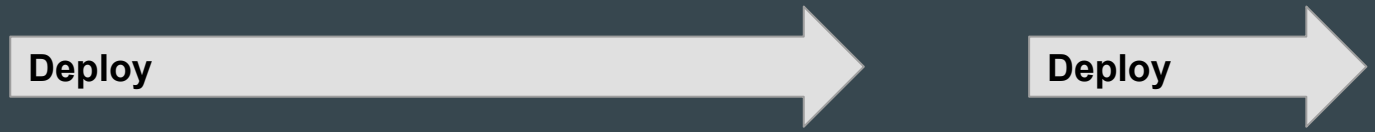
Staging

Siste stopp før applikasjonen blir "offentlig"

Programvaren er antatt stabil

Produksjon

Applikasjonen vår er offentlig tilgjengelig for andre - den versjonen som "alle andre" ser



“Her er maskinen min”

“Her er maskinen min”

Containeren er identisk hele veien - det at vi har et bilde som sier “her er applikasjonen med alle avhengigheter i gitte versjoner som kjører i dette miljøet” betyr at det er samme innhold som kjører i alle miljø.

Det som før var “Det virker på min maskin” er byttet ut med “alle maskiner er like, så funker det her så funker det der”

En viktig begrensning!

Ettersom containerene våre ikke kan endres permanent - så har de ikke varig lagring av data! Når en container restartes er det akkurat som om den aldri har blitt kjørt tidligere!

Varige lagringsbehov løses ved at en container får tilgang til “ekstern” lagring - harddiskplass på maskinen som kjører containeren, isolert (eller sammen med, avhengig av konfigurasjon) fra andre containere - men denne er *ikke en del av innholdet i containeren* og konfigureres spesielt

Versjonskontroll

Dette gir samme maskin hver gang!

```
# base image (python)
FROM alpine:3.12.0

# Kopièr inn filene fra lokal katalog til /app
COPY appenvaar /app

# Tjenesten vår kjører på port 4242
PORT 4242

# Kjør kommandoen “startappen”
CMD [“startappen”]
```

Denne definisjonen legger vi i

Git / versjonskontroll

Slik at hvis noen endrer definisjonen av containeren vår, kan vi se *hva* som ble endret, *hvorfor* det ble endret og se historikken

Eksempel - applikasjonen vår har fått tildelt nytt portnummer (dvs, hvilken port / hvilket nummer den kan kontaktes på)

```
ENV APP=daichi:app
ENV DAICHI_SETTINGS=/app/production.cfg
-ENV PORT=${PORT:-9822}

ENV LC_ALL=en_US.UTF-8
ENV LC_LANG=en_US.UTF-8
@@ -26,7 +26,7 @@ ADD . /app
# without being an actual dependency
RUN pip install -e .

-EXPOSE ${PORT:-9822}

ENTRYPOINT /usr/local/bin/gunicorn -w 5 -b :$PORT $APP
```

```
2
3   ENV APP=daichi:app
4   ENV DAICHI_SETTINGS=/app/production.cfg
5  +ENV PORT=${PORT:-9014}
6
7   ENV LC_ALL=en_US.UTF-8
8   ENV LC_LANG=en_US.UTF-8

26  # without being an actual dependency
27  RUN pip install -e .
28
29 +EXPOSE ${PORT:-9014}
30
31  ENTRYPOINT /usr/local/bin/gunicorn -w 5 -b :$PORT $APP
32
```

.. og noen fant ut at vi burde starte applikasjonen på en annen måte for å jobbe rundt et problem i et bibliotek

2 Dockerfile View file

@@ -28,5 +28,5 @@ RUN pip install -e .

```
28
29  EXPOSE ${PORT:-9014}
30
-31 ENTRYPOINT /usr/local/bin/gunicorn -w 5 -b :$PORT $APP
32
```

```
28
29  EXPOSE ${PORT:-9014}
30
+31 ENTRYPOINT /usr/local/bin/gunicorn -w 3 -k gthread --threads 2
   -b :$PORT $APP
32
```

Og hvis noen lurer på hvorfor, så forklarer commitmeldingen vår grunnen til at endringen ble gjort!

Vi får alle fordelene vi er vant med fra versjonskontroll av kode

Vi kan rulle tilbake endringer til noe vi VET fungerer!

Vi kan spore **HVEM** som gjorde endringen, **HVA** som ble endret, **HVORFOR** den ble gjort og **NÅR** den ble gjort

Vi kan branche - gjøre prøveendringer uten å ødelegge for hovedversjonen vår

Vi kan koordinere endringer fra mange personer inn i samme beskrivelsesfil - på samme måte som vi kan for kode!

(hvis dere ikke har fått det med dere enda, så er de fleste som driver med data glad i at ting er versjonskontrollert)

Og vi kan integrere med CI/CD-pipelinen vår for å bygge containerne våre - og så deploye dem senere hvis vi vil

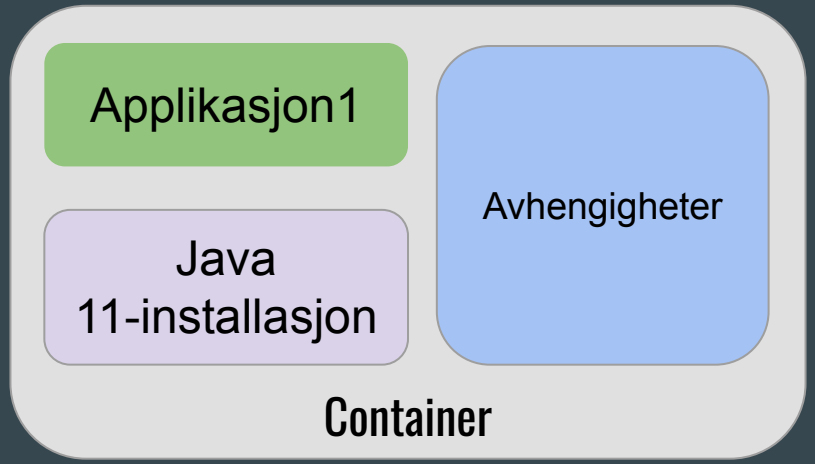
GitHub Actions + programmatisk definert oppsett

```
name: test-and-publish-image
on:
  push:
    branches:
    - main
jobs:
  tests:
    uses: ./.github/workflows/tests.yaml
  publish:
    name: publish
    needs: [tests]
    runs-on: ubuntu-latest
```

GitHub Actions + programmatisk definert oppsett

```
steps:
  - name: Set up Docker Buildx
    uses: docker/setup-buildx-action@v2
  - name: Login to Docker Hub
    uses: docker/login-action@v2
    with:
      username: ${{ secrets.DOCKER_HUB_USERNAME }}
      password: ${{ secrets.DOCKER_HUB_ACCESS_TOKEN }}
  - name: Build and push
    uses: docker/build-push-action@v4
    with:
      push: true
      tags: <orgname>/${{ github.event.repository.name }}:v1.${{
github.run_number }}
```

Og når vi nå har denne...



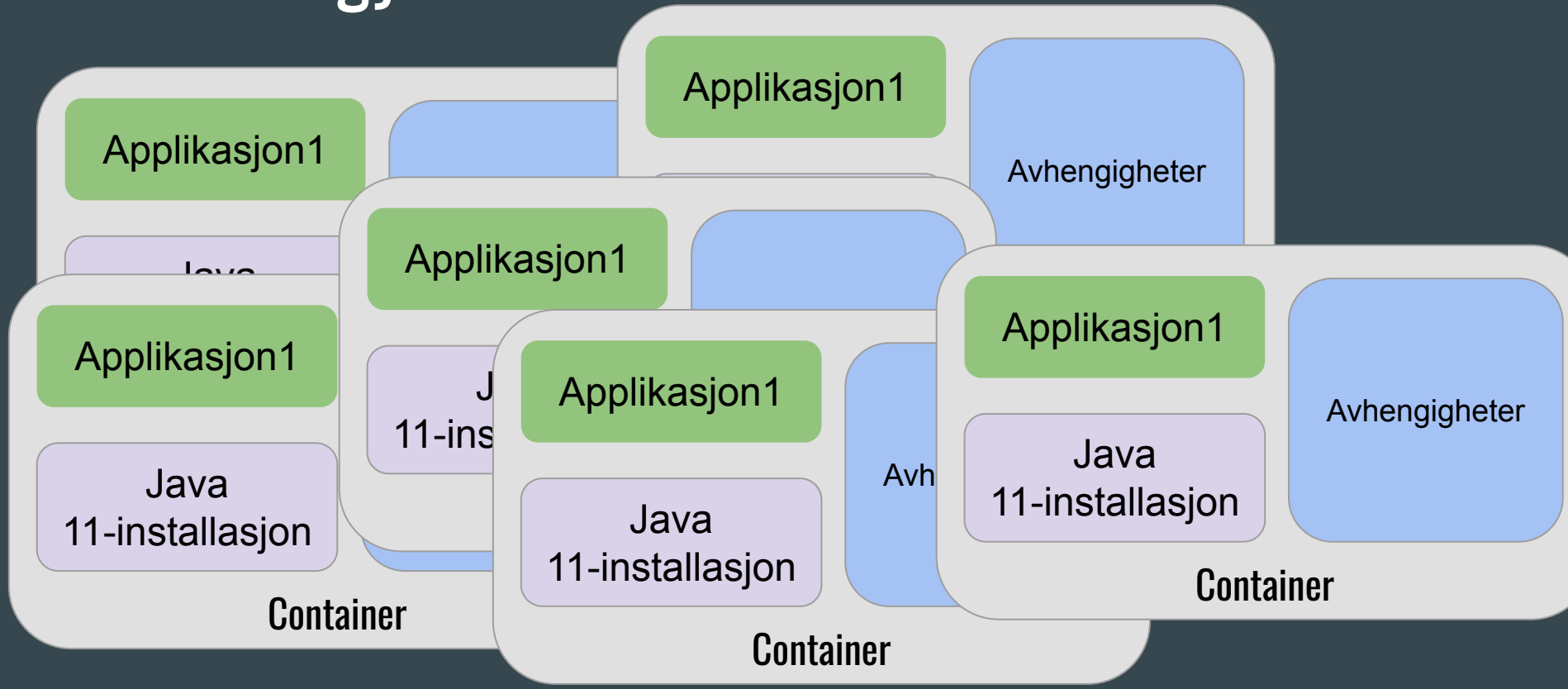
Og plutselig får disse

Photo by Rob Curran on Unsplash

Kilde: CC-SA-BY / torkildr, flickr

Og egentlig trenger disse

Så begynner vi å snakke om orchestration



Videotips

Live-eksempel med Kubernetes og Docker

Bygging og deploying av containere i skyen

Typiske feil når man skriver tester