

Orchestration

Mikrotjenester, Kubernetes og Serverless



Mats Lindh, mats.lindh@gmail.com

<https://cat.devnull.no/orchestration-2024.pdf>

Microsoft finds underwater datacenters are reliable, practical and use energy sustainably

10 hard-to-swallow truths they won't tell you about software engineer job

```

    }
}
$service = $this->serviceBackend->findOneById($session->service_id);
$isTextChat = ($service->type == Service::TYPE_TEXT_CHAT);
if (!$isTextChat) {
    $url = [
        '/dashboard/meeting-room',
        'sessionId' => $session->session_id,
        'sessionType' => $session->classletter
    ];
    $this->layout = false;
    /*
    * Edge has a known bug where the user camera rotates by 90 degrees,
    * when (1) we redirect,
    * (2) we open a new tab on _blank and
    * (3) we use the internal camera.
    *
    * This specific problem happens when certain email clients insert a
    * `target="_blank"` into reminder emails.
    *
    * Fuck Edge.
    * Relates to https://developer.microsoft.com/en-us/microsoft-edge/platform/issues/1476984
    */
    return $this->render('/edge-routing', ['url' => Url::to($url)]);
}
$this->layout = 'dashboard/chat';

$contractSession = $this->findOrCreateContractSession($session);
$expert = $this->expertBackend->findOneById($session->expert_id);
$client = $this->customerBackend->findOneById($session->client_id);
$textChatMessages = Chat::findAll(['session_id' => $session->session_id]);

```

DAGENS PROBLEM

Hvordan håndterer vi ulike behov
for trafikk og for drift av tjenestene
og applikasjonene våre?

Dagens plan

- Mikrotjenester
- Orchestration
 - (nei, ikke symfoni-typen)
 - Hva i alle dager
 - Hvorfor i alle dager
 - Infrastructure as a Service
 - Kubernetes

Mikrotjenester

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.

Hotellbestillingssystem

```
graph TD; A[Hotellbestillingssystem] --> B[Statistikk]; A --> C[Ordr]; B --> D[orderAttempt()]; B --> E[orderSuccess()]; C --> F[medlemskap()];
```

Statistikk

orderAttempt()

orderSuccess()

Ordr

medlemskap()

Det kan hende det gir mening
å flytte dette ut til en egen
tjeneste / applikasjon

Hotellbestillingssystem

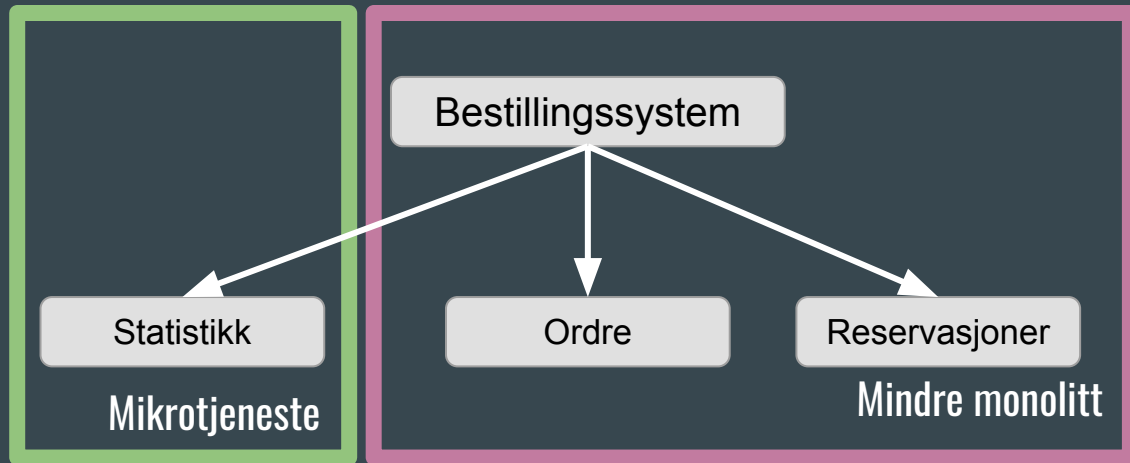
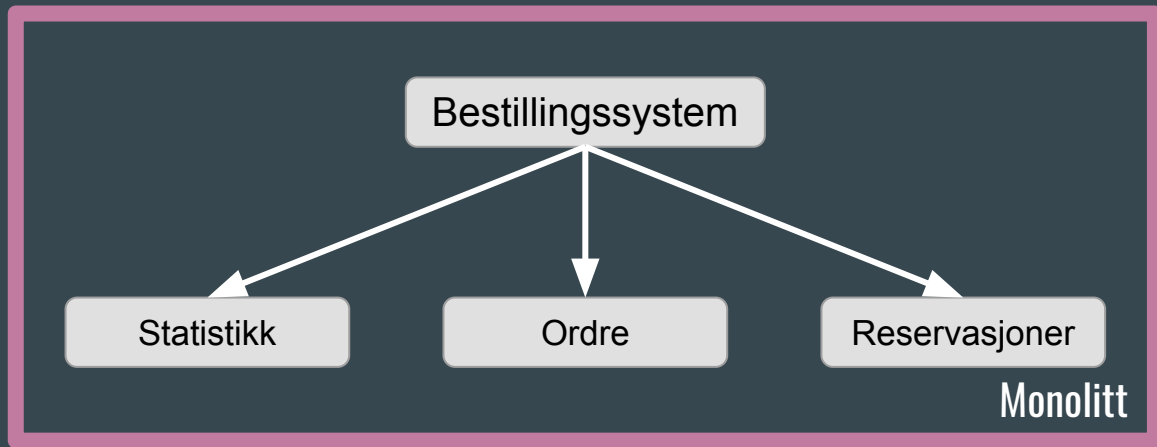
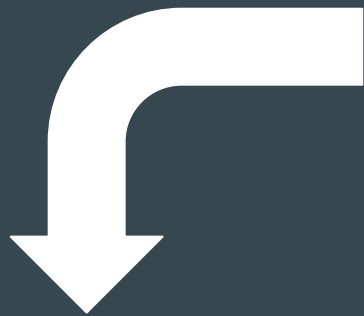
```
graph TD; A[Hotellbestillingssystem] --> B[Statistikk]; A --> C[Ordre]; A --> D[Reservasjoner];
```

Statistikk

Ordre

Reservasjoner

Monolitt



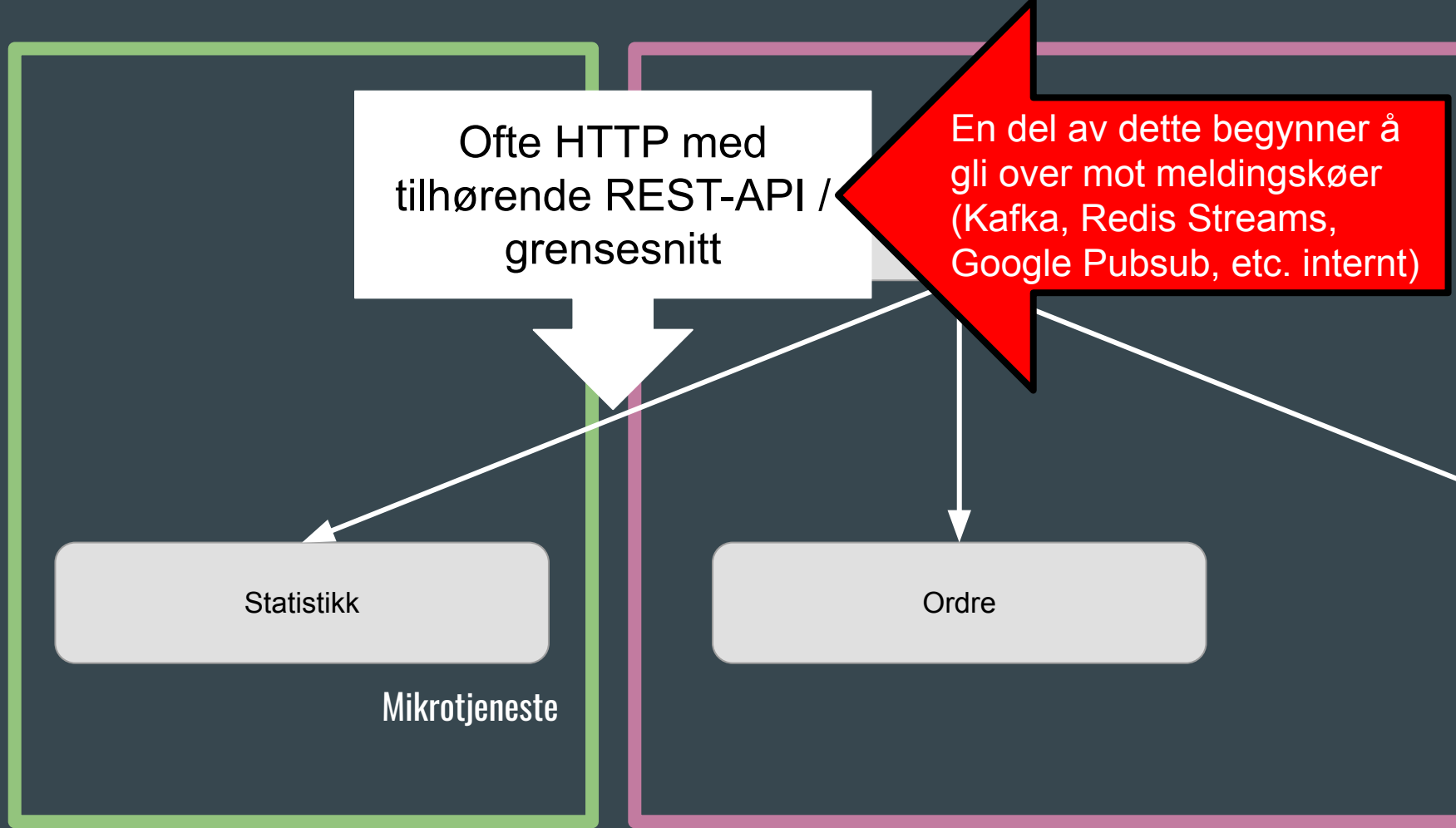
Ofte HTTP med
tilhørende REST-API /
grensesnitt

En del av dette begynner å
gli over mot meldingskøer
(Kafka, Redis Streams,
Google Pubsub, etc. internt)

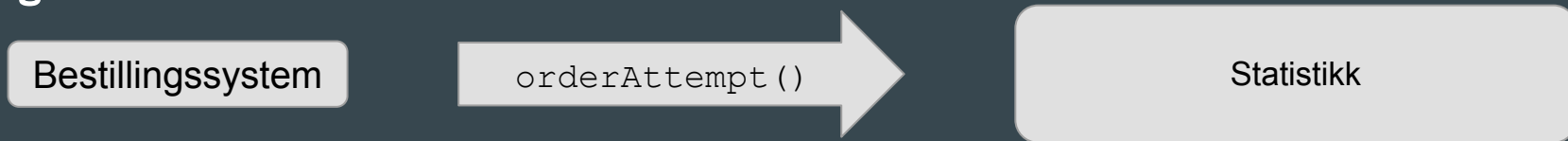
Statistikk

Mikrotjeneste

Ordre



Tidligere



Med en mikrotjeneste



Bestillingssystem

Mailutsending

Hotellvisning

Hotellsystem

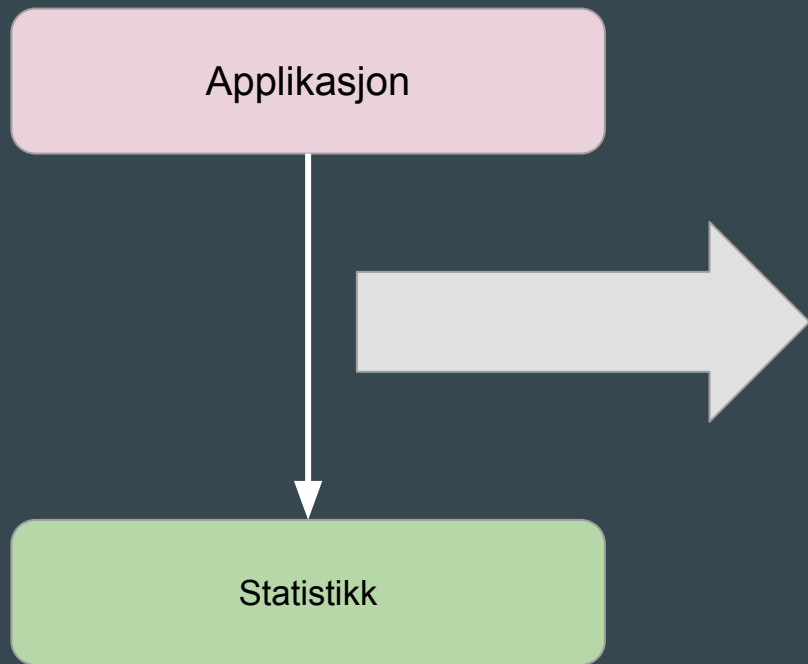
HTTP
POST /events
{"type": "order-atten

HTTP
POST /events
{"type": "mai

HTTP
POST /event
{"type":

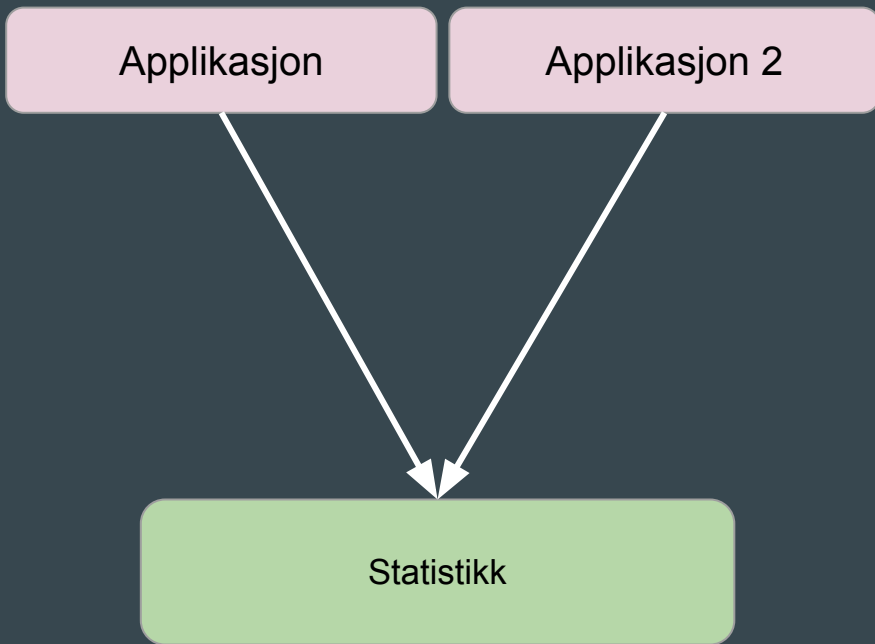
HTTP
POST /events
{"type": "checked-in"}

Statistikk



Begrensninger

- API-et er nå et offentlig API - endringer i Statistikk kan ikke forvente at Applikasjon oppdateres
- Bruddet må skje i brytningen mellom distinkte ansvarsområder / domener - Statistikk er en egen greie som er uavhengig av hva applikasjonen gjør og driver med, og kan logisk abstraheres bort og gjenbrukes



Mål

- Applikasjon 1 og Applikasjon 2 kan begge benytte seg av statistikk uten å ha noe annet forhold til tjenesten enn API-et, og kan deployes og oppdateres uavhengig av Statistikk
- Statistikk kan deployes og utvides uten at Applikasjon og Applikasjon 2 oppdateres
- Teamet som har ansvaret for Statistikk kan være helt uavhengig av de andre teamene



ORGANISATORISK SKILLE

Separasjonen er viktig - dette gir oss autonome team og tjenester som kan oppgraderes og driftes uavhengig av applikasjonene som bruker dem

Applikasjon

Applikasjon 2

Statistikk

Brukerdatabase

Applikasjon 3

Nyhetsbrev

Ved gjennomført lagdeling i koden til
Applikasjon vil det være “trivielt” å bytte ut
Statistikk-delen med en tjeneste i bakgrunnen
i stedet

Mikrotjenester blir en veldig eksplisitt lagdeling - den delen av koden har sitt helt eget ansvar, måte å fungere på og bruk av teknologi - det eneste som er tilgjengelig for “utenforstående” er det offentlige API-et til tjenesten

Vi kan fullstendig endre implementasjonen
inne i **Statistikk**, så lenge API-et fortsatt har
samme semantiske mening - og få fordelene i
alle applikasjonene som benytter tjenesten
samtidig

Statistikk-tjenesten har fått for mye data og det er et ønske om å flytte datalagringen over til Googles BigTable?

Implementasjonsdetaljer.

Eneste som betyr noe for applikasjonene som bruker tjenesten!

Statistikk m/API

InfluxDB?

BigTable?

es?

Noen har funnet ut at Go gir mer mening for Statistikk-tjenesten og tester viser at det øker ytelsen med 400% over Visual Basic .NET?

Implementasjonsdetaljer.
(.. bare sørg for at API-et er det samme)

Men - som vanlig - pass på. Det gir sjelden mening å starte med mikrotjenester (som vi alle gjorde da vi hørte om begrepet og at dette var det nye, fancy!), men heller flytte ut funksjoner til separate tjenester

når behovet melder seg

**Distribuerte systemer er, av mangelen på et bedre ord,
et helvete å debugge.**

**Det å følge en request gjennom tre-fire-ti-tolv relaterte
systemer for å finne ut hvor feilen oppstår er noe du ønsker å
gjøre minst mulig av..**

**.. særlig dersom tjenesten bare har én applikasjon som
bruker den, og den like gjerne kunne ha vært inne i
applikasjonen og tilgjengelig rett i debuggeren din.**

Perfect is the enemy of good

Gold Plating

Gold plating is the phenomenon of working on a project or task past the point of diminishing returns. For example: after having met the requirements, the project manager or the developer works on further enhancing the product, thinking the customer will be delighted to see additional or more polished features, rather than what was asked for or expected.

Nedetid kan bli dyrt, så sørg for at
applikasjonen er feiltolerant

.. *innenfor hva som er nyttig for applikasjonen.*

Det er alltid mer penger
og mer tid på prosjekt.




Et prosjekt som *aldri blir lansert* har imidlertid
enda mindre nytte.

**.. men i noen tilfeller er det
en sentral del av jobben din**

Black Friday 2019









Black Friday er i gang! Se kampanjen her!

POWER Bedrift > POWER Support > Fotofremkalling > Logg inn MyPower >

POWER Hva leter du etter?  KUNDESERVICE  FINN POWER-BUTIKK  HANDLEKURV

Hvitevarer Hjem og fritid Velvære Kjøkkenutstyr Mobil Data Datakomponenter Gaming TV Lyd Foto Kjøkken **BLACK FRIDAY**

BLACK FRIDAY

BLACK FRIDAY Er du klar?	 110 Hvitevarer	 66 Hjem & husholdning	 64 Lyd & bilde	 96 PC, nettbrett & gaming	 53 Mobil, trening & foto	 15 Kjøkken	 191 Smarte hjem	 Kundeavis	 Se alle
------------------------------------	--	---	--	---	---	--	---	--	--

BLACK FRIDAY HOS POWER
power.no kl. 20:00. I butikk fra fredag 07:00

Faksimilie: power.no



Shop Black Friday deals



Hi, Mats

Customer since 2002

Recommendations for you



Your Orders



Electronics

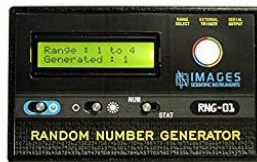


Computers &
Accessories



Home & Kitchen

Recently viewed



Sat, Nov 23

[See your browsing history](#)

Save big on toys



[Shop now](#)

Holiday Deals on Home



[Shop now](#)



VELKOMMEN TIL BLACK FRIDAY HOS ELKJØP

Akkurat nå er det mange kunder som ønsker å gjøre et kjøp hos oss og vi jobber på spreng med å få unna køen. Vi sender deg til elkjo.no så snart vi har mulighet. Takk for at du venter. Dersom det er veldig mange som forsøker å gjennomføre handelen samtidig så kan du oppleve å havne i flere køer.

[Hva er dette?](#)

Du er på websiden om: 16 minutter



● Oppdatert: 09:38:44

Ke-ID: [79bbba-40e2-4010-af1b-27a412775093](#)

QUEUE-IT

BLACK FRIDAY!

Innenriks

Elkjøp gikk glipp av 50 millioner etter Black Friday-fadese

– Trykket var langt over det vi hadde ventet, sier Elkjøp Norge-toppsjef Fredrik Tønnesen.

DN +



1 min

Publisert: 02.12.19 – 14.50

Oppdatert: 31 minutter siden



Komplett Group

11,376 milliarder omsatt på nett i 2023

31 millioner/dagen

1,29 millioner/timen

21.6k/minuttet

360kr./sekundet

Orchestration

Infrastructure as a Service

Amazon May Deployment Stats

(production hosts & environments only)

11.6 seconds

Mean time between deployments (weekday)

1,079

Max # of deployments in a single hour

10,000

Mean # of hosts simultaneously receiving a deployment

30,000

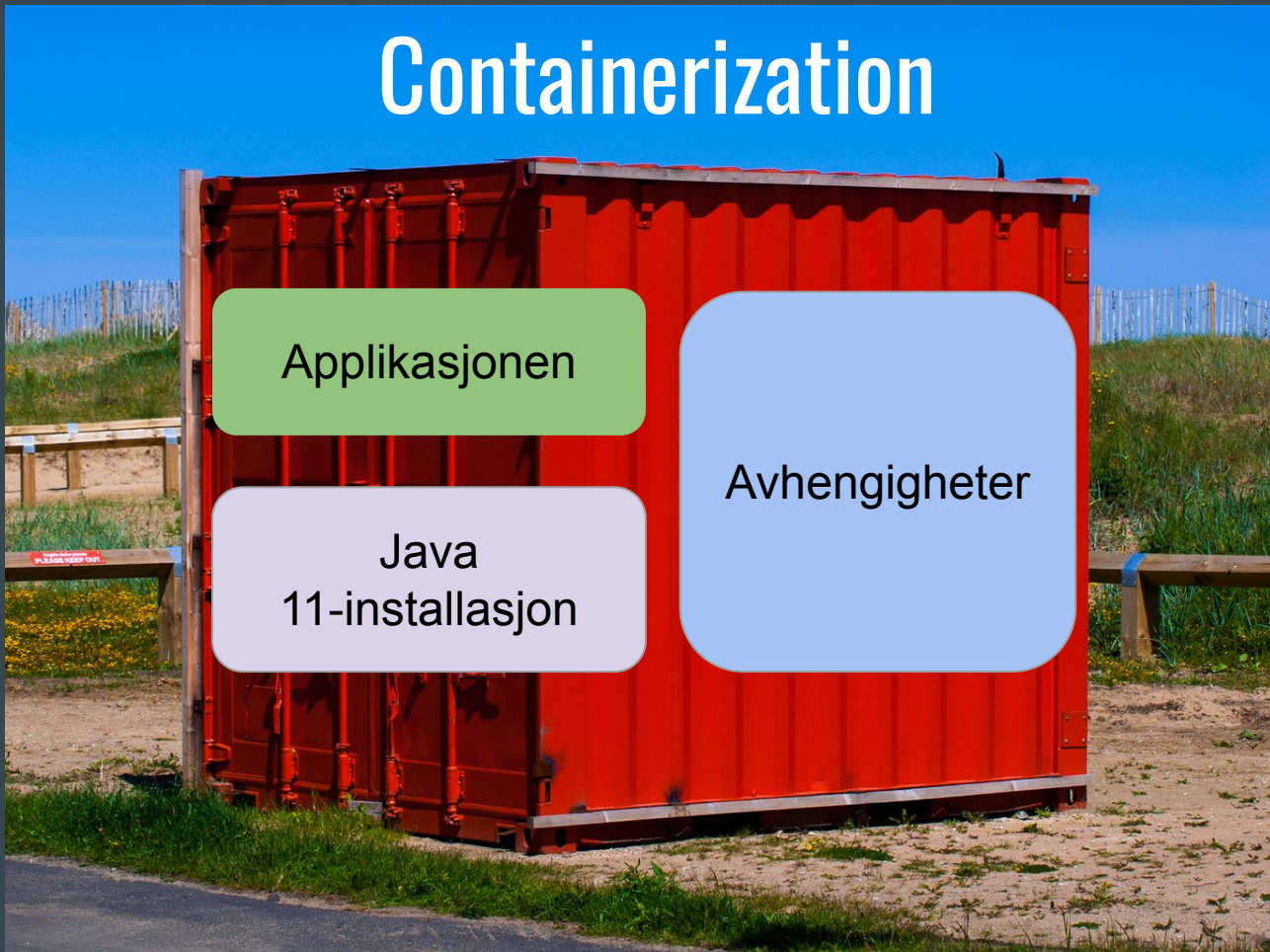
Max # of hosts simultaneously receiving a deployment

Containerization

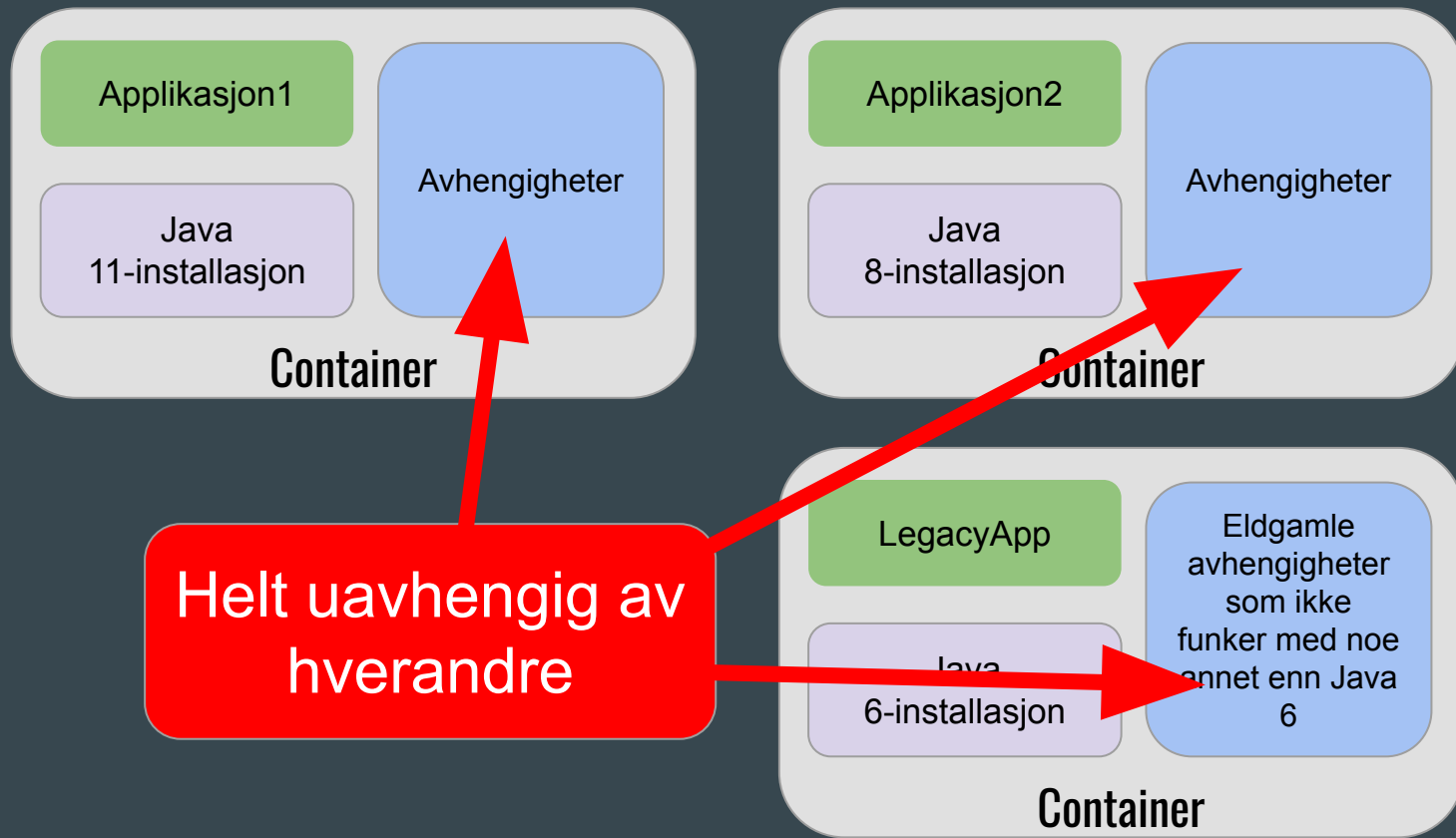
Applikasjonen

Java
11-installasjon

Avhengigheter



Isolasjon og stabilitet



“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 gjør det lett å kopiere og overføre til andre steder.

En viktig begrensning!

Ettersom containerne 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!

Vi har definert miljøet til
applikasjonen som et sett med
regler (`Dockerfile`) i stedet for å
gjøre statisk konfigurasjon og
oppsett av fysiske servere..

Og vi har stappet alt inn i en container



.. så kan vi begynne å automatisere
“hvor mange containere trenger
vi” og “hvor store skal
containerene våre være?”

Mens containerization handler om å *dytte hele applikasjonen vår inn i en passende liten container* - så handler orchestration om *koordineringen* av disse containerene

Container 1
Superapp

Container 2
Superapp

Container 3
Superapp

Container 4
Superapp

Container 5
Superapp

Container 6
Superapp

Container 7
Superapp

Container 8
Superapp

Container 9
Superapp

Container 10
Superapp

Container 11
Superapp

Container 12
Superapp

**Containerene sprer vi utover
serverene våre ... eller til Google /
Microsoft / Amazon / etc.**

Orchestration-plattform

Nettverk
(kryptering,
mellom noder,
etc.)

Lagring
(permanent,
midlertidig)

Applikasjoner
(containere)

Ingress/Egress
(Trafikk inn/ut)

etc.

etc.

etc.

Orchestration-plattform

Dette er målet vårt - det andre rundt er infrastruktur for å håndtere applikasjonene - det som gir oss verdi!

Applikasjoner
(containere)

Ingress/Egress
(Trafikk inn/ut)

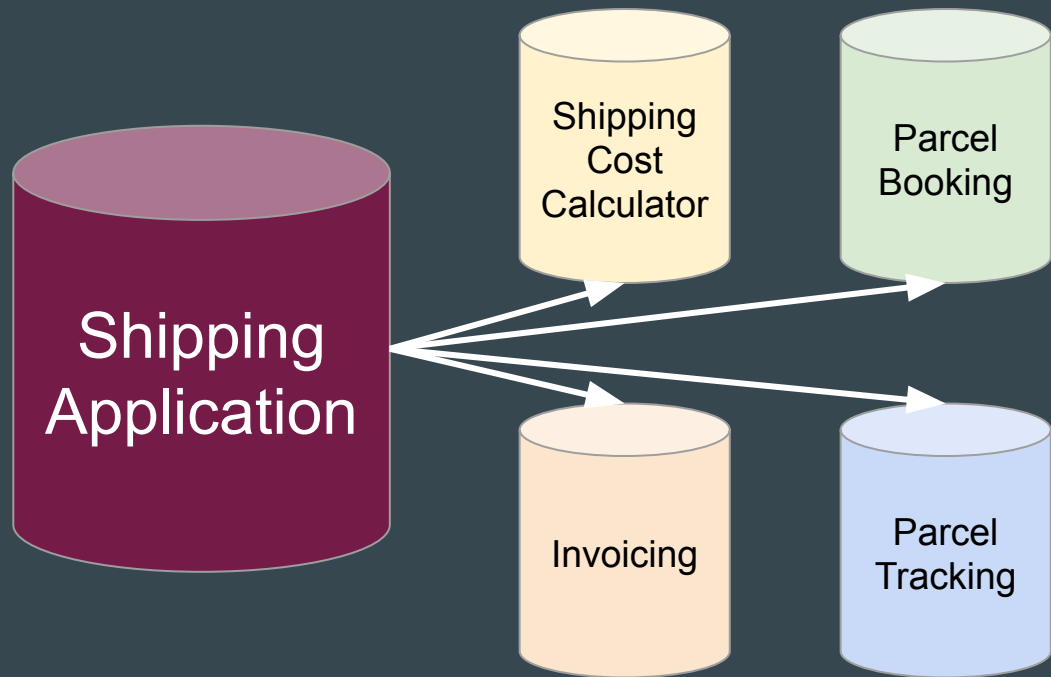
etc.

etc.

etc.

“Men kunne vi ikke bare ha hatt 12 store servere og kjørt Superapp direkte på disse - i stedet for å styre med containere?”

Det kunne vi! Men vanligvis har vi ikke bare en applikasjon - vi har gjerne mange - og særlig hvis vi bygger ting som mikrotjenester!



You've got crates, we've got rates!

Container 1
Shipping Application

Container 2
Shipping Cost Calculator

Container 3
Parcel Booking

Container 4
Invoicing

Container 5
Parceltracking

Alle kan kjøre på samme fysiske eller virtuelle server (eller på helt forskjellige - det spiller ingen rolle her), men er isolert slik at de kun ser sin egen applikasjon og egne avhengigheter!

Nå som vi har applikasjonen og miljøet til applikasjonen som en container, så har vi muligheten til å justere antallet instanser av hver tjeneste opp og ned etter hvert som behovet melder seg, slik at vi bruker maskinvaren vår **mest mulig fleksibelt og effektivt -- og billigst mulig!**

Container 1
Shipping Application

Container 4
Shipping Application

Container 7
Shipping Application

Container 2
Shipping Cost Calculator

Container 5
Shipping Cost Calculator

Container 8
Shipping Cost Calculator

Container 3
Parcel Booking

Container 6
Parcel Booking

Container 9
Invoicing

Container 10
Parceltracking

Container 11
Parceltracking

Container 12
Parceltracking

Etter hvert er Shipping Cost Calculator ute av bildet og vi trenger ikke lenger ressursene som er tilordnet disse containerene

**Da stopper vi containerene som inneholder
“Shipping Cost Calculator”**

Container 1
Shipping Application

Container 2
Shipping Cost Calculator

Container 3
Parcel Booking

Container 4
Shipping Application

Container 5
Shipping Cost Calculator

Container 6
Parcel Booking

Container 7
Shipping Application

Container 8
Shipping Cost Calculator

Container 9
Invoicing

Container 10
Parceltracking

Container 11
Parceltracking

Container 12
Parceltracking

Ettersom tjenestene ikke lenger er i bruk kan disse ressursene nå automagisk brukes av de andre (og av nye) containerene

Men som vanlig - utviklere og driftere er late personer
.. så dette vil vi gjøre mest mulig automagisk uten at vi
trenger å tenke så mye på det - vi vil bare si at vi vil ha
flere eller færre noder, og at hele “serverparken” vår (..
containerparken?) med avhengigheter kan beskrives

.. og når vi kan beskrive noe, så
kan vi legge det i ...


VERSJONSKONTROLL!

**Ettersom container-runtimen vår
(containerd, docker, etc.)
styrer containerene våre, kan den også styre
hva slags ressurser hver container blir tildelt**


.. og når vi kan beskrive hva slags maskinvare som skal tildeles containerene våre, så betyr det at vi kan **versjonskontrollere** hva applikasjonen vår skal ha tilgang til av **maskinvare!**

myapp-eployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  namespace: mynamespace
spec:
  selector:
    matchLabels:
      app: myapp
  replicas: 2
  template:
    metadata:
      labels:
        app: myapp
```



```
spec:
  containers:
    - name: myapp
      image: myorg/mycontainer:
      ports:
        - containerPort: 8080
      envFrom:
        - secretRef:
            name: mysecrets
      livenessProbe:
        httpGet:
          path: /
          port: 8080
          initialDelaySeconds: 3
          periodSeconds: 30
      imagePullSecrets:
        - name: mycredentials
```



Eksempel fra Amedia

- templatet ut fra det som vanligvis ligger i definisjonene

Hvilken container?

Hvilken versjon?

`image=../daichi:0.0.3`

`replicas=1` Antall instanser som skal kjøres

`cpu.limit=500m` Maks CPU-andel

`memory.limit=500Mi` Maks minne-andel

`cpu.request=200m` Tildelt CPU-andel

`memory.request=100Mi` Tildelt minne-andel

Trenger applikasjonen mer minne? Mer CPU?


```
image=../daichi:0.0.3  
replicas=1  
cpu.limit=500m  
memory.limit=500Mi  
cpu.request=200m  
memory.request=100Mi
```

```
image=../daichi:0.0.3
```

```
replicas=1
```

```
cpu.limit=1
```

<- Maks CPU opp fra 0.5

```
memory.limit=2Gi
```

<- Maks minne opp
fra 500MB

```
cpu.request=200m
```

```
memory.request=100Mi
```

**Enklere (og raskere) å håndtere enn å ringe
Dell for å få tak i mer minne til den maskinen
dere kjøpte for åtte år siden**

**Når vi har versjonene vi har releaset og
versjonskontrollerte definisjoner, så kan vi
også rulle tilbake endringene våre**

La vi ut en ny versjon som ikke fungerte?

**Bytt tilbake til den gamle
containeren!**

Hvilken versjon?



```
image: myorg/mycontainer:v0.2.1
```

vs

```
image: myorg/mycontainer:v0.2.2
```

Trenger vi flere instanser av applikasjonen vår?

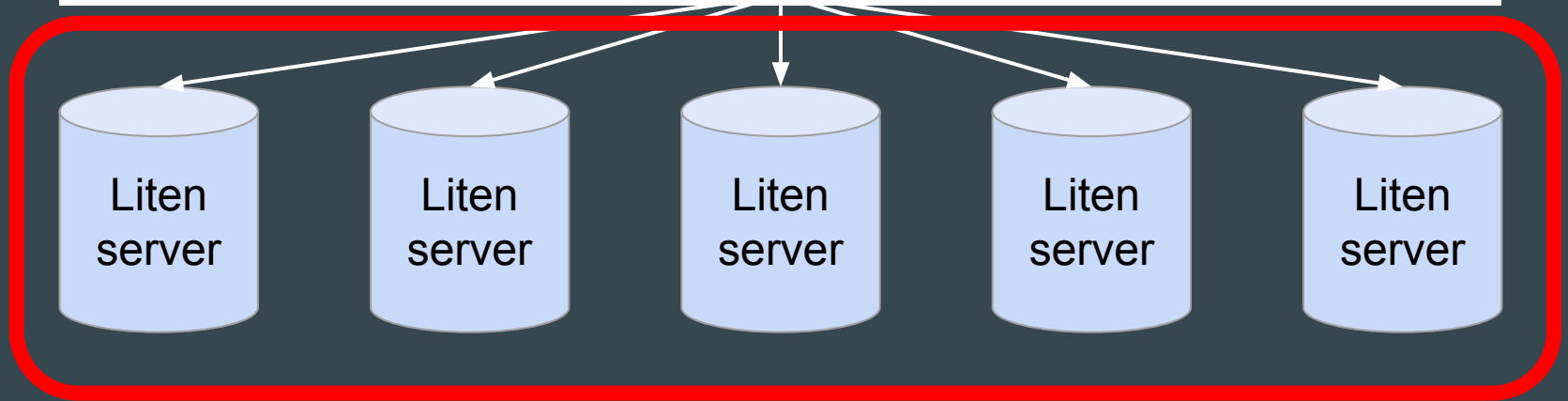
Antall instanser som skal kjøre

```
replicas: 2
```


replicas: 3

← 3

Hele dette her som noen måtte
sette opp og installere med
riktige versjoner tilsvarer nå ..



```
replicas: 5
```

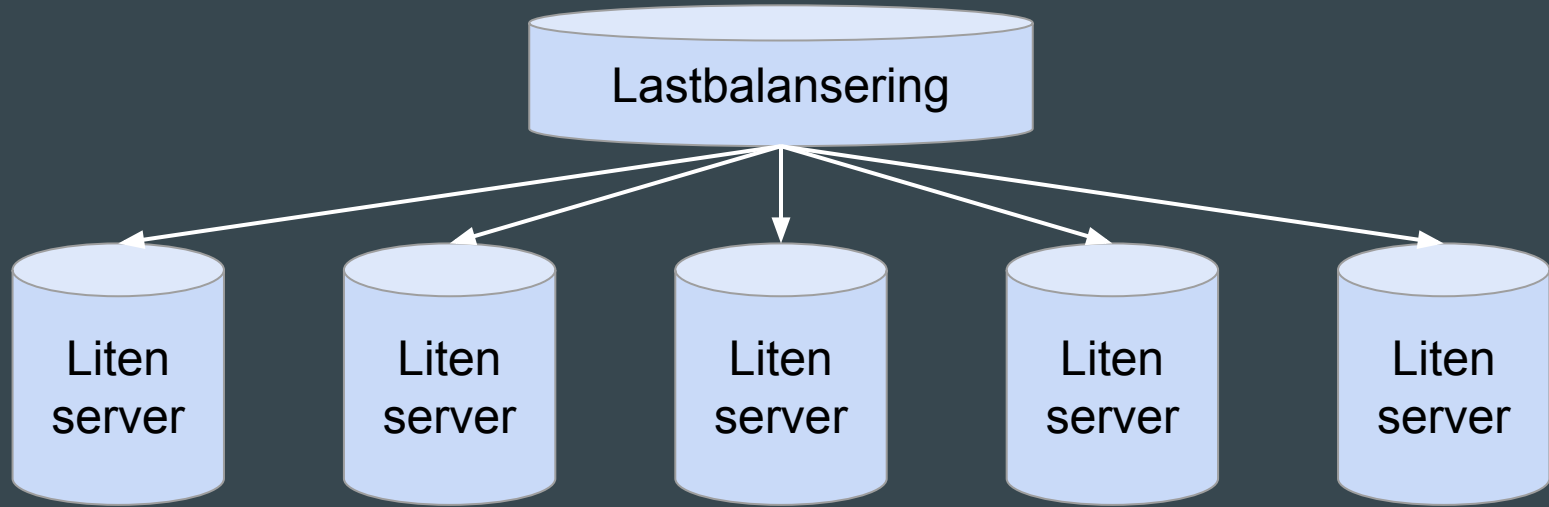
**“Vi vil ha fem instanser av
denne containeren”**

Oppgradering i fart

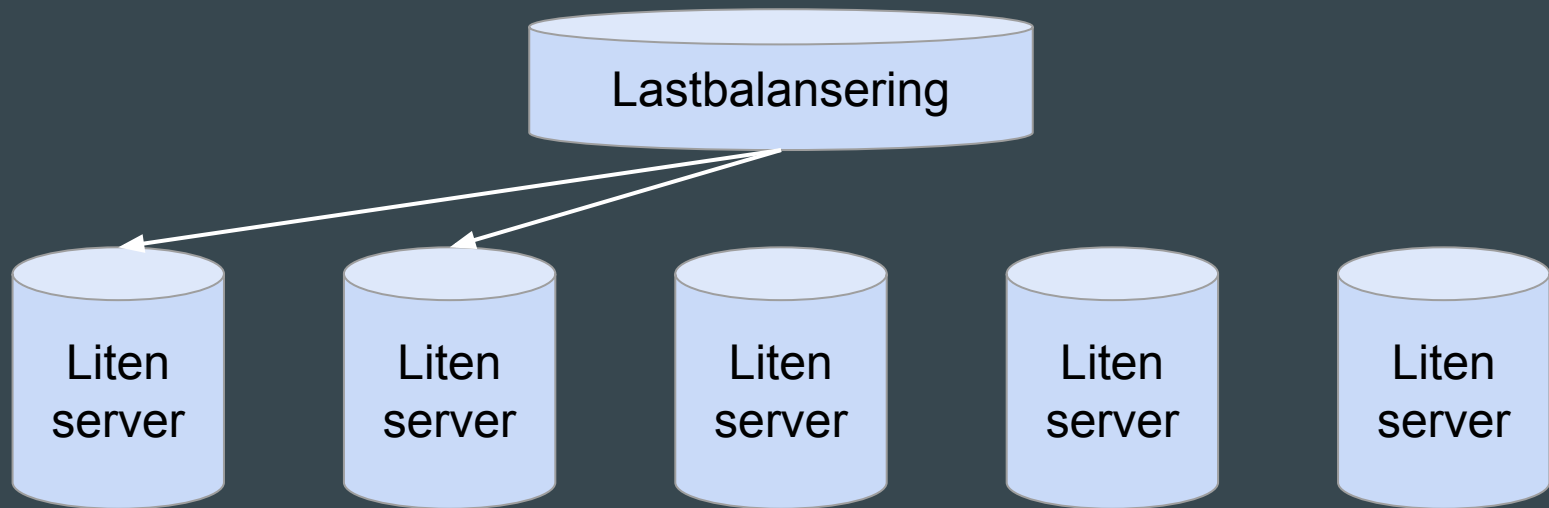
Når vi har en *løst oppdelt arkitektur basert på mikrotjenester*, så kan vi oppgradere disse “i fart” - det vil si uten at vi får noen merkbar nedetid for brukerne av tjenestene våre.

Hver tjeneste kan oppgraderes separat og uavhengig av de andre, og om vi har skrevet tjenestene våre til å kunne spres horisontalt, kan vi ved å bruke lastbalansering eller caching (for kjente ressurser) sørge for at klientene får svar selv om deler av tjenesten vår er nede

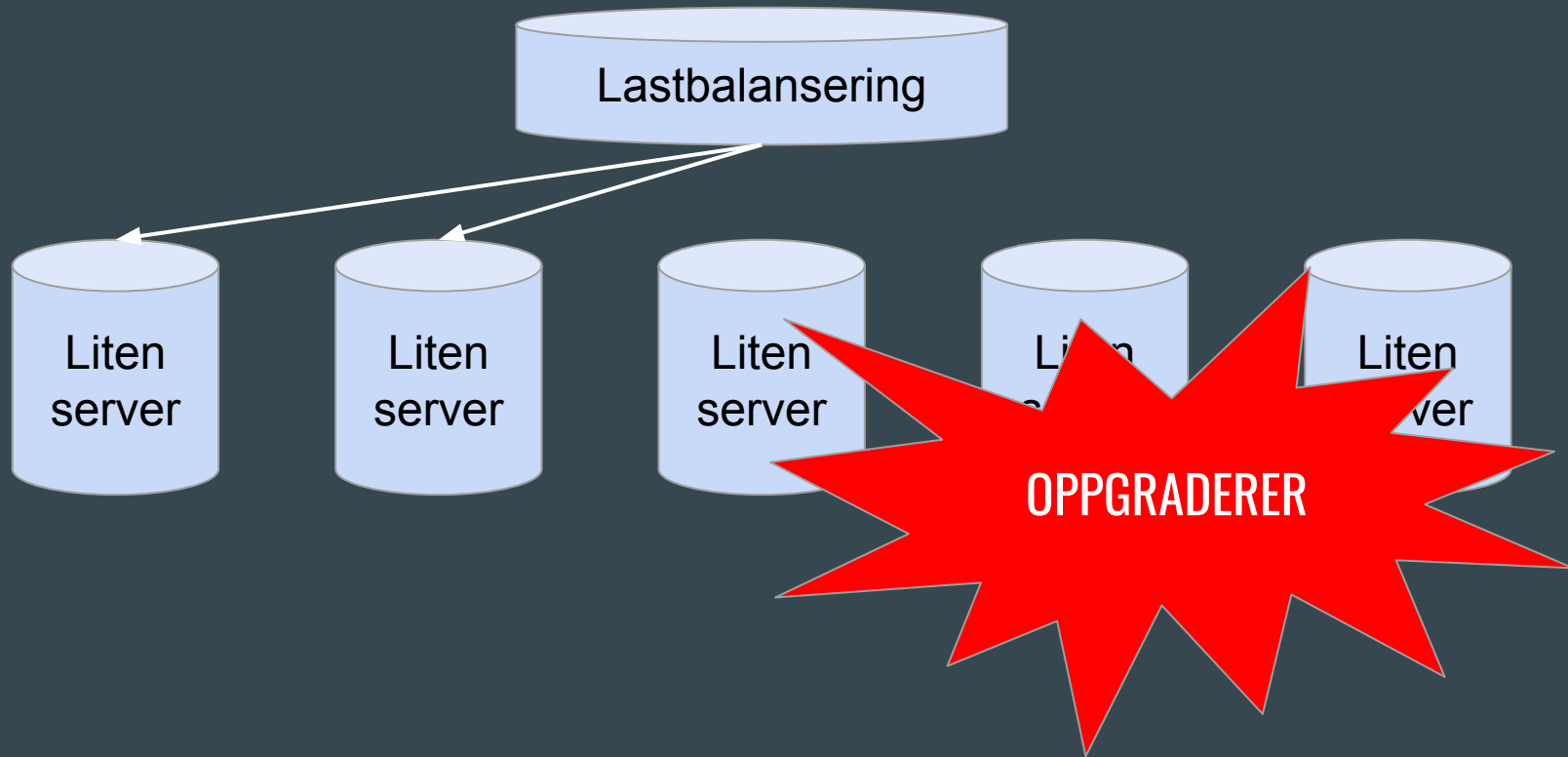
Tradisjonell manuell metode for oppgradering i fart



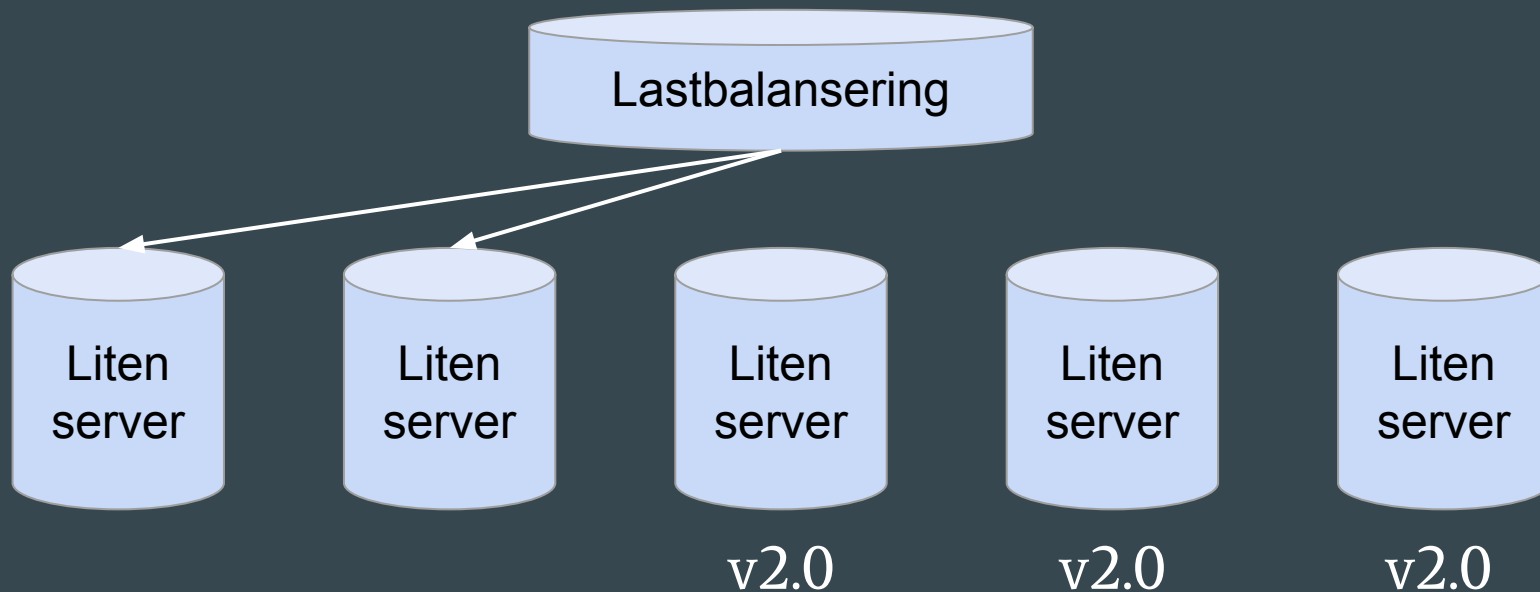
Vi fjerner serverene vi skal oppgradere fra dem som aktivt får trafikk



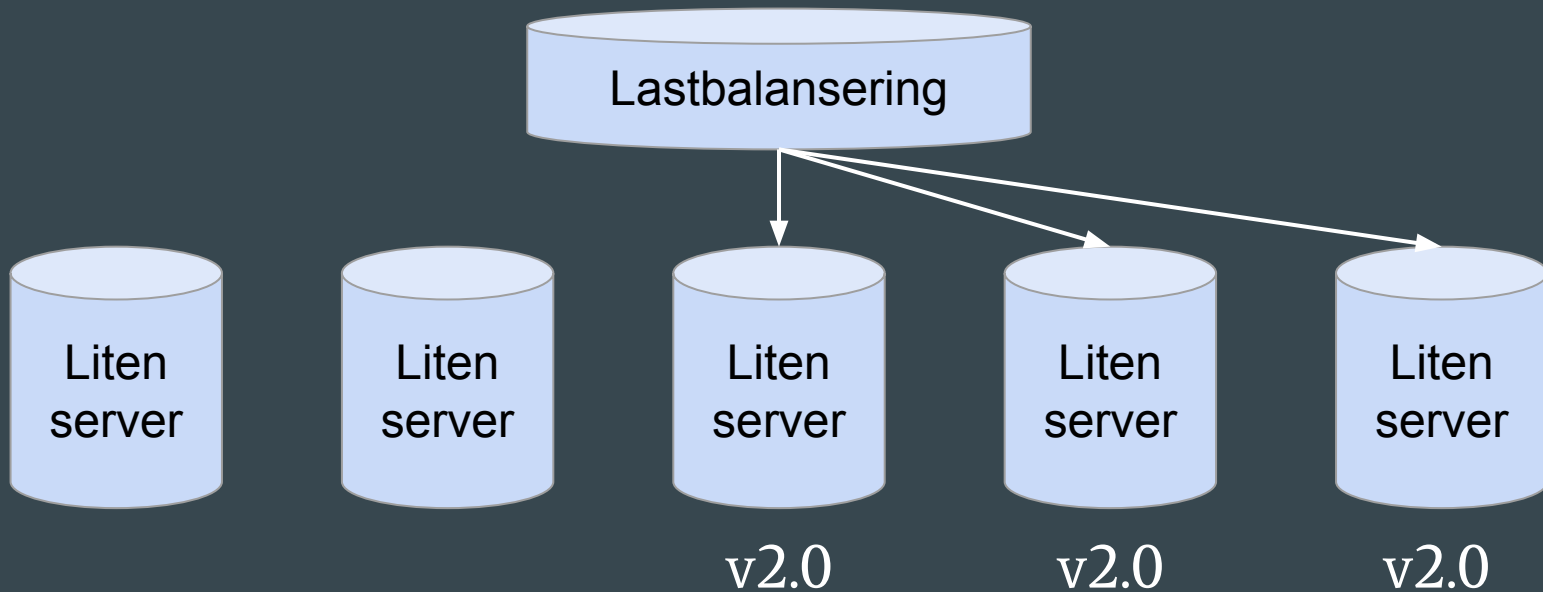
.. oppgraderer dem som ikke får trafikk ..



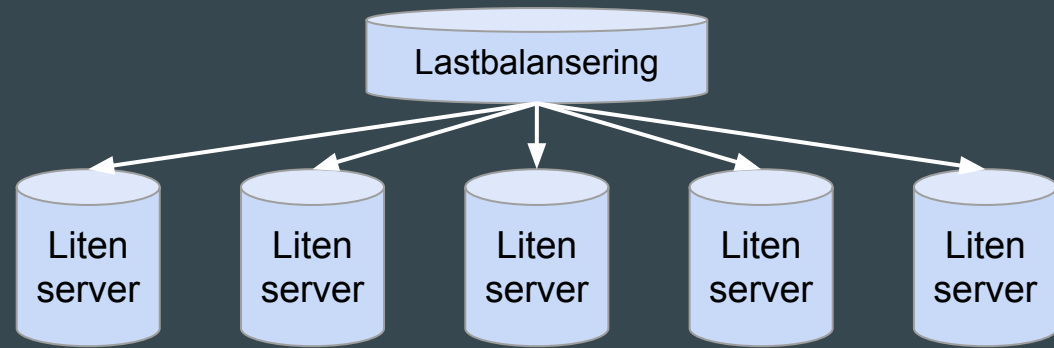
Ny versjon på plass!



.. og flytter all trafikken vår over til de nye, oppgraderte serverene

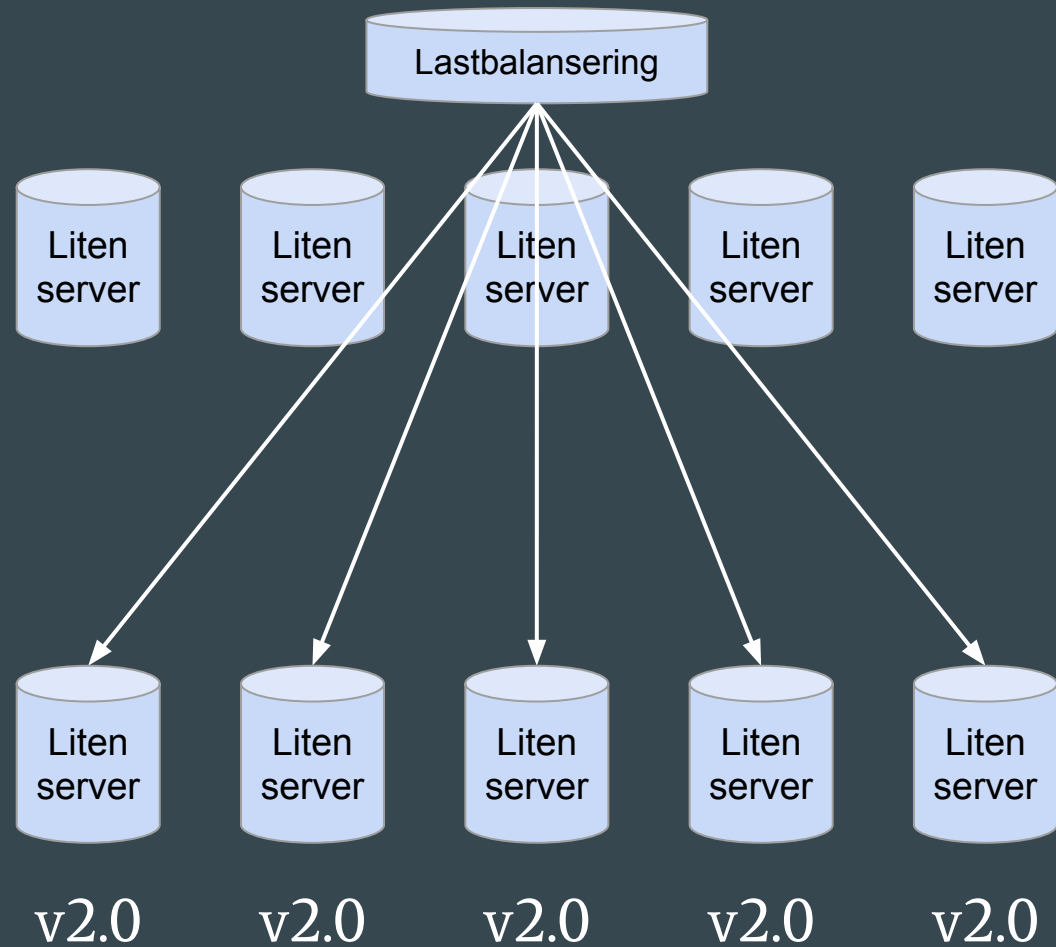


Dette kan nå systemet *gjøres helt automagisk*
for oss - ettersom systemet har “maskinene”
våre (containerene) og vet hvordan de skal
startes og stoppes



Dette gjøres vanligvis ved å spinne opp nye containere parallelt med den kjørende applikasjonen, og så flytte trafikken over når alle de nye containerene er klare





Dette gjøres vanligvis ved å spinne opp nye containere parallelt med den kjørende applikasjonen, og så flytte trafikken over når alle de nye containerene er klare

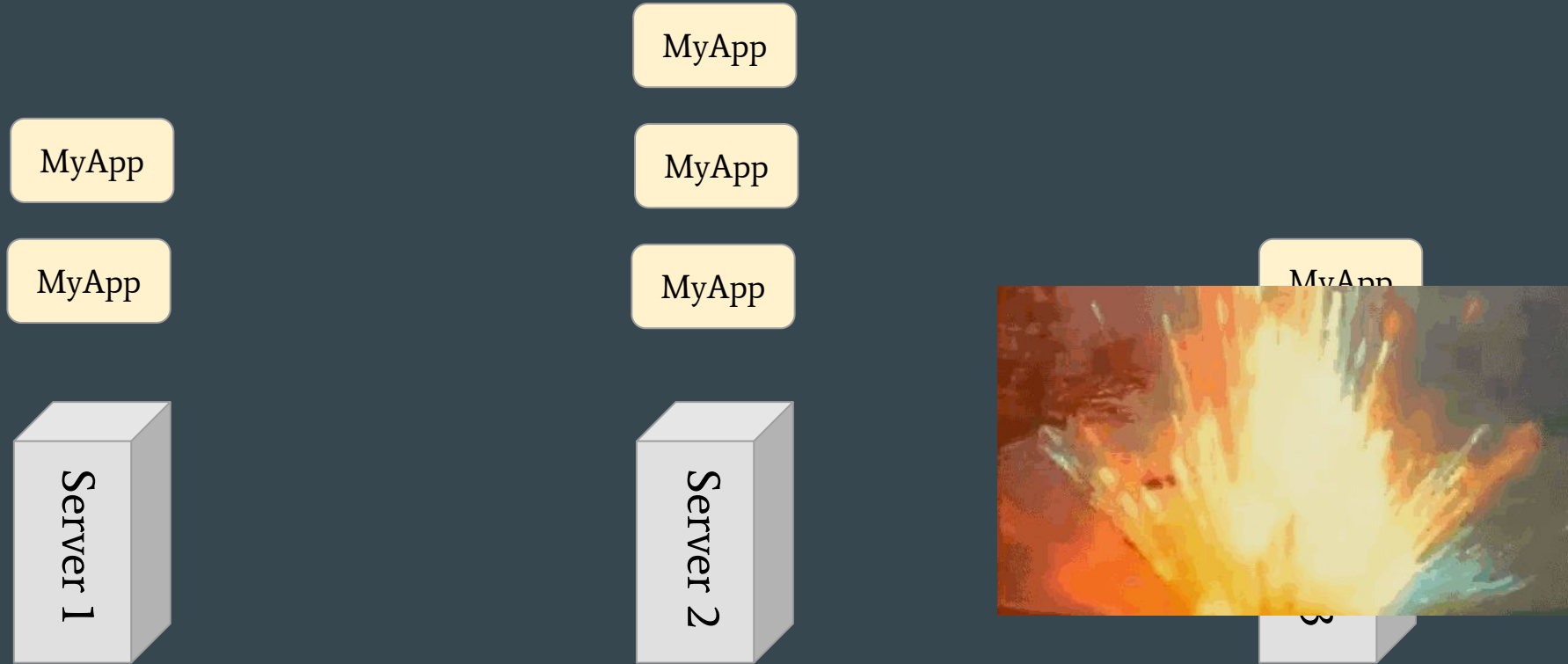
Kilde: New Zealand Defence
Force from Wellington, New
Zealand

A large container ship is shown listing heavily to its starboard side in the ocean. The ship is covered in multi-colored shipping containers. A helicopter is visible on the water's surface near the ship's bow. The image is used as a metaphor for server downtime.

Hvis en av serverene våre dør
akkurat når det passer oss minst

Så kan systemet oppdage at “oi, nå mangler vi ...” og starte opp containerne på en annen, tilgjengelig node

Selvbalsensering



Dette er ikke teoretisk!



Tilsvarende kan vi håndtere andre, lignende situasjoner automagisk

Containerne våre kjører alle med 80% belastning? *Start flere automagisk.*

Containerne våre kjører med 5% belastning? *Stopp alle som er til overs og la andre containere bruke ressursene*

Konseptet med at vi har hele infrastrukturen
vår tilgjengelig som *konfigurasjon* - og ikke
som fysisk håndtering av servere og manuell
installasjon - har selvsagt fått sitt eget navn

Infrastructure as a Service

Nå kan vi ha datasenteret vårt i
VERSJONSKONTROLL!

Og vi kan i teorien flytte det til en annen tilbyder ved å laste opp konfigurasjonen vår og si hvor containerene våre kan hentes fra

Kubernetes (k8s)

Smooth sailing with Kubernetes



SwiftOnSecurity
@SwiftOnSecurity



One time I tried to explain Kubernetes to someone.

Then we both didn't understand it.

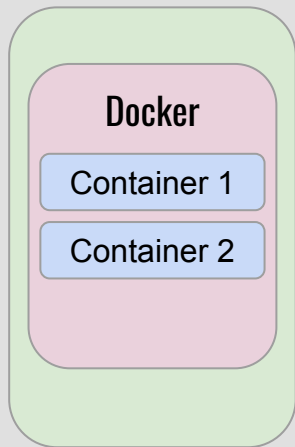
16:40 · 06/08/2019 · [Twitter for iPhone](#)

Kubernetes er det mest brukte Orchestration-systemet i dag

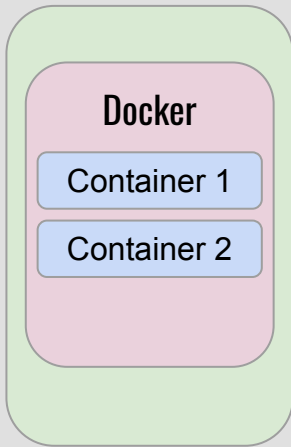
Kubernetes koordinerer containerene våre ut over serveren våre, og sørger for at de kjører i riktig antall, har tilgang til riktige ressurser (f.eks. diskplass) og er tilgjengelige på nettverket

Kubernetes

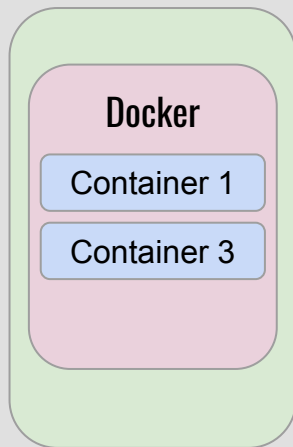
Kubernetes Node



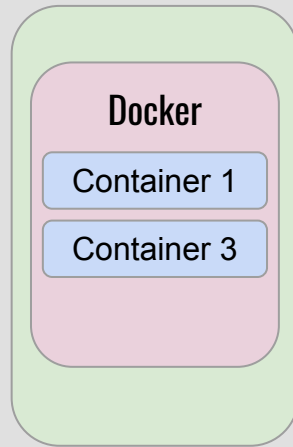
Kubernetes Node



Kubernetes Node



Kubernetes Node



Kubernetes har et sett med glimrende eksempler for dem som vil prøve selv - rett i nettleseren

Eller på egen maskin med minikube

Anbefaling: Learning-K8S: Keep it Simple

k3s: “Situations where a PhD in K8s clusterology is infeasible”

Og når du først har et API til clusteret og
datasenteret ditt ...

Xlskubectl

*“Your scientists were so preoccupied with whether or not they
could, they didn’t stop to think if they should.”*

Ingress

Det eksterne (utenfor clusteret) inngangspunktet til Kubernetes. Typisk implementert som en lastbalanser / webtjener. I egne oppsett er dette ofte basert på nginx/traefik/caddy (automagisk/bortgjemt for deg).

Har ekstern IP.

Deployment

Styrer antallet pods som skal være tilgjengelig for hver container.

Pods som kjører en gitt container identifiseres med `labels`.

Pod

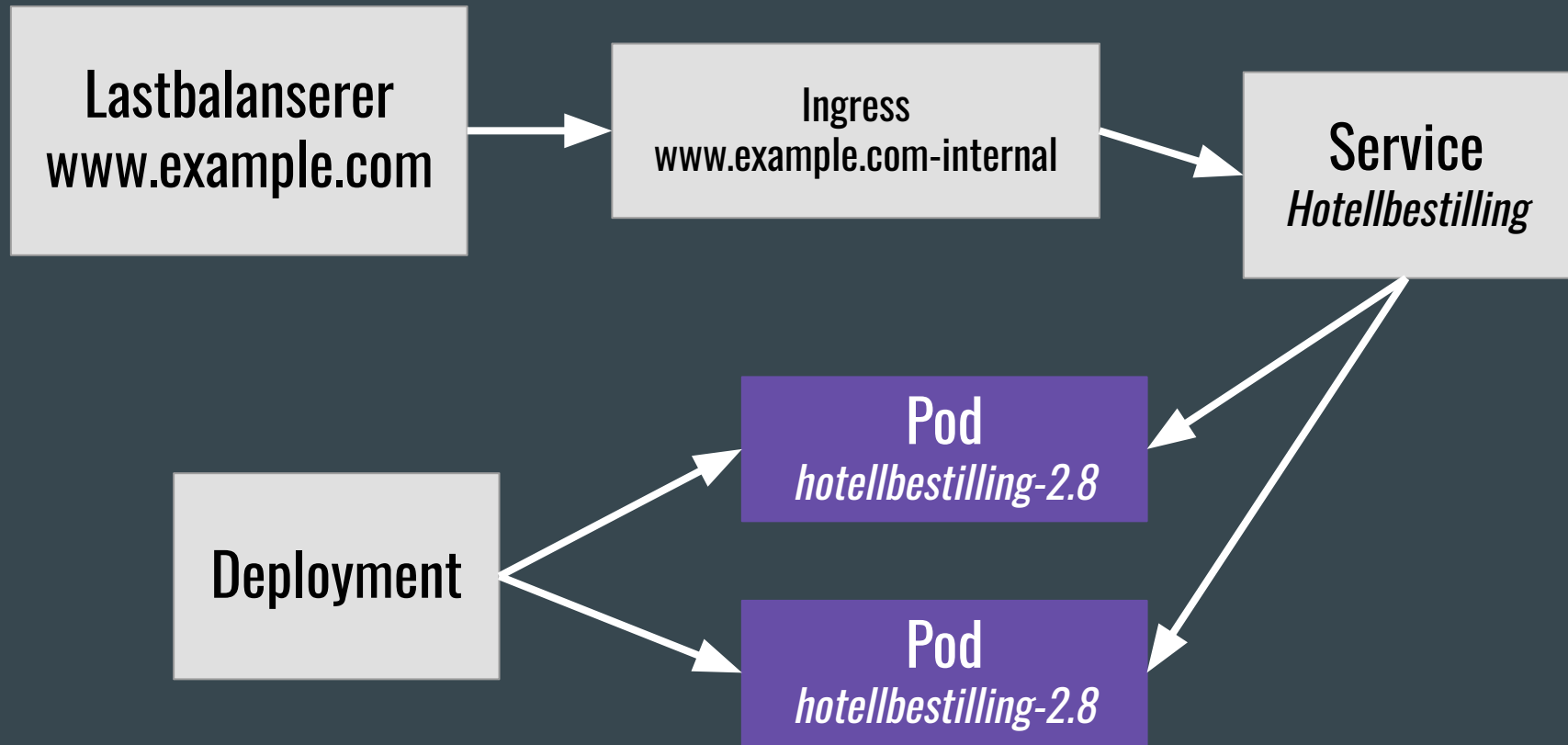
Et sett med kjørende containere.

Service

Den interne (i k8s-clusteret) abstraksjonen av “disse podsene utgjør denne servicen” - når pods dør/drepes/startes får de nye interne ip-adresser, og servicen er abstraksjonen som gjemmer dette bort og last-balanserer mellom pods

(forenklet, sjekk [Kubernetes-dokumentasjonen for komplett oversikt](#))

Kubernetes-struktur



Control plane - backendet som styrer alt

Eksempel med Kubernetes for en enkel app

Live-eksempel fra forelesningen H2020

se-hiof-docker-example

Bruk minikube for å leke med dette selv!

Nå har vi definisjonen av applikasjonslaget i datasenteret vårt i versjonskontroll. Neste steg?

La oss få inn definisjonen av ressursene som datasenteret skal bestå av i versjonskontroll også!

Terraform - Pulumi - etc.

```
resource "google_compute_instance" "vm_instance" {  
  name          = "terraform-instance"  
  machine_type  = "f1-micro"  
  
  boot_disk {  
    initialize_params {  
      image = "debian-cloud/debian-9"  
    }  
  }  
  
  network_interface {  
    network = google_compute_network.vpc_network.name  
    access_config {  
    }  
  }  
}
```

[Kilde: Terraform - Create resource / Google Cloud](#)

**Dette definerer den faktiske “hardwaren”:
nettverket, lagringsmengder, etc. som ligger
under Kubernetes og andre tjenester, slik at
også ressursbruken hos leverandøren er
abstrahert bort til utviklernivå**

Serverless og Cloud Functions

“Serverless allows you to build and run applications and services without thinking about servers. It eliminates infrastructure management tasks such as server or cluster provisioning, patching, operating system maintenance, and capacity provisioning.”

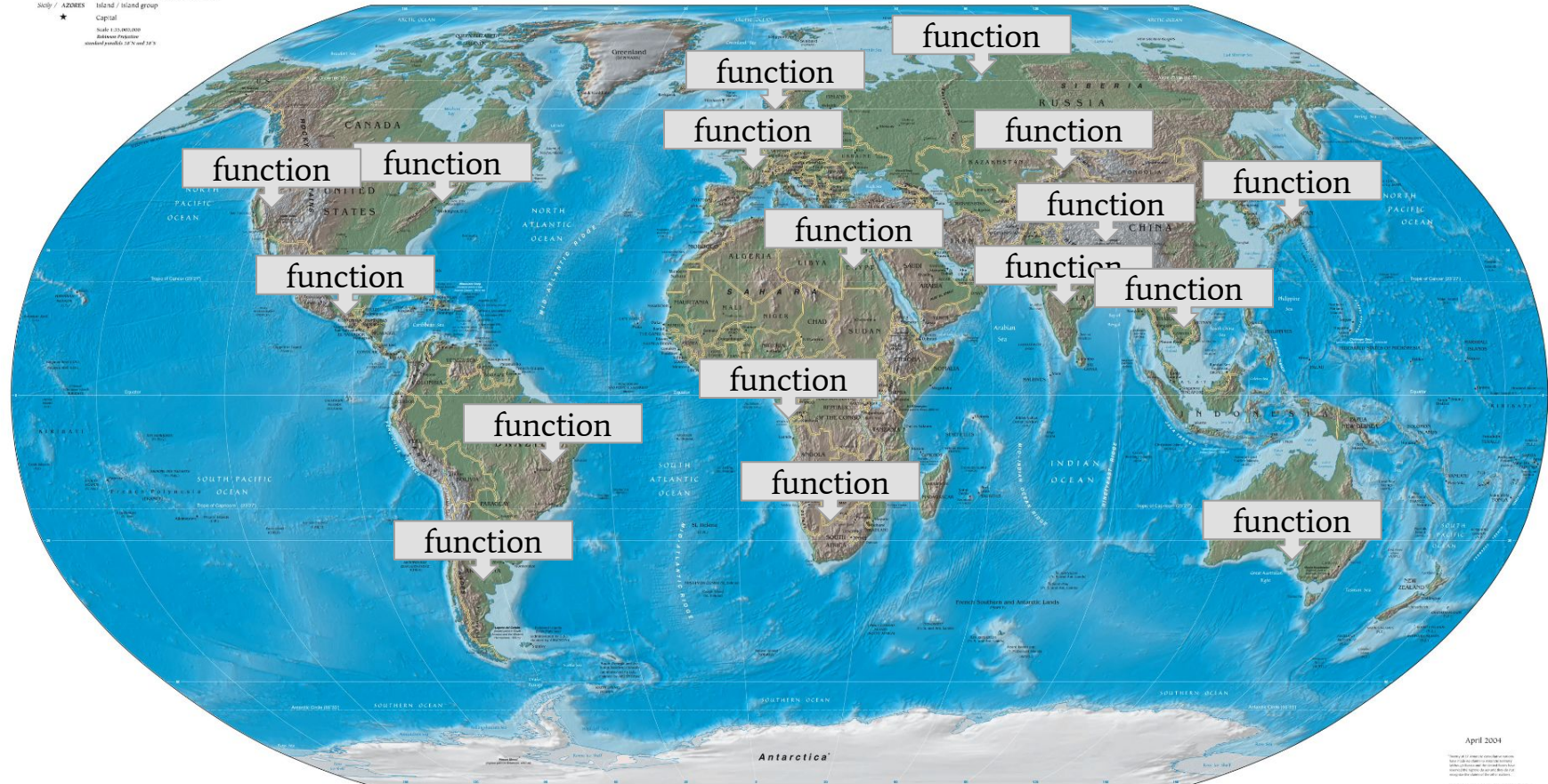
```
async function handleRequest(request) {  
    // kode  
}
```

I stedet for å si “her er ‘maskinen’ du skal kjøre”, så sier vi “her er den lille kodesnutten du skal kjøre når noen går til <url>”

Physical Map of the World, April 2004

AUSTRALIA
Bermude
Søstø / A.D. 2005
★
Capital
Scale 1:10,000,000
Reference: 10°N and 10°S

.. og denne funksjonen kan vi distribuere rundt verden.



Kilde: CIA World Factbook

Serverless m/Cloud Run

*“Cloud Run is a serverless offering from Google which allows you to host docker containers which **will run whenever triggered through an API** .”*

It abstracts away all infrastructure management such as provisioning, configuring, and managing servers, so you focus only on writing code. It automatically scales up and down from zero depending on traffic almost instantaneously, so you don't have to worry about scale configuration ever. Cloud Run also charges you only for the resources you use (calculated down to the nearest 100 milliseconds), so you will never have to pay for your over-provisioned resources.

Det Store
Verdensomspennende
Internettet

.. men jeg har
ingen kjørende..

Cloud Run

GET /blogposts

Hotellbestilling-2.8

Response
HTTP/1.1 200 OK

```
[{"name": "Grand  
Hotell", ...}]
```



Tips: sett spending limits MED EN GANG



sophie ✓
@netcapgirl



she said "take me somewhere expensive"



9:45 PM • Sep 4, 2024

How WebSockets cost us \$1M on our AWS bill

Eksempler

Bygging av Docker Container + Deploying til fly.io og
Google Cloud Run

Serverless med Cloudflare Workers

Application Deployment and Testing Strategies
(Google Cloud Architecture Center)