Kubernetes - en rask intro

Kristian Lyngstøl, Storo, Mars 2018

Agenda

- Intro
- Grunnkonsepter
- Lagring
- Nettverk
- Avansert-ish
- Feilsøking
- Diskusjon

Containere

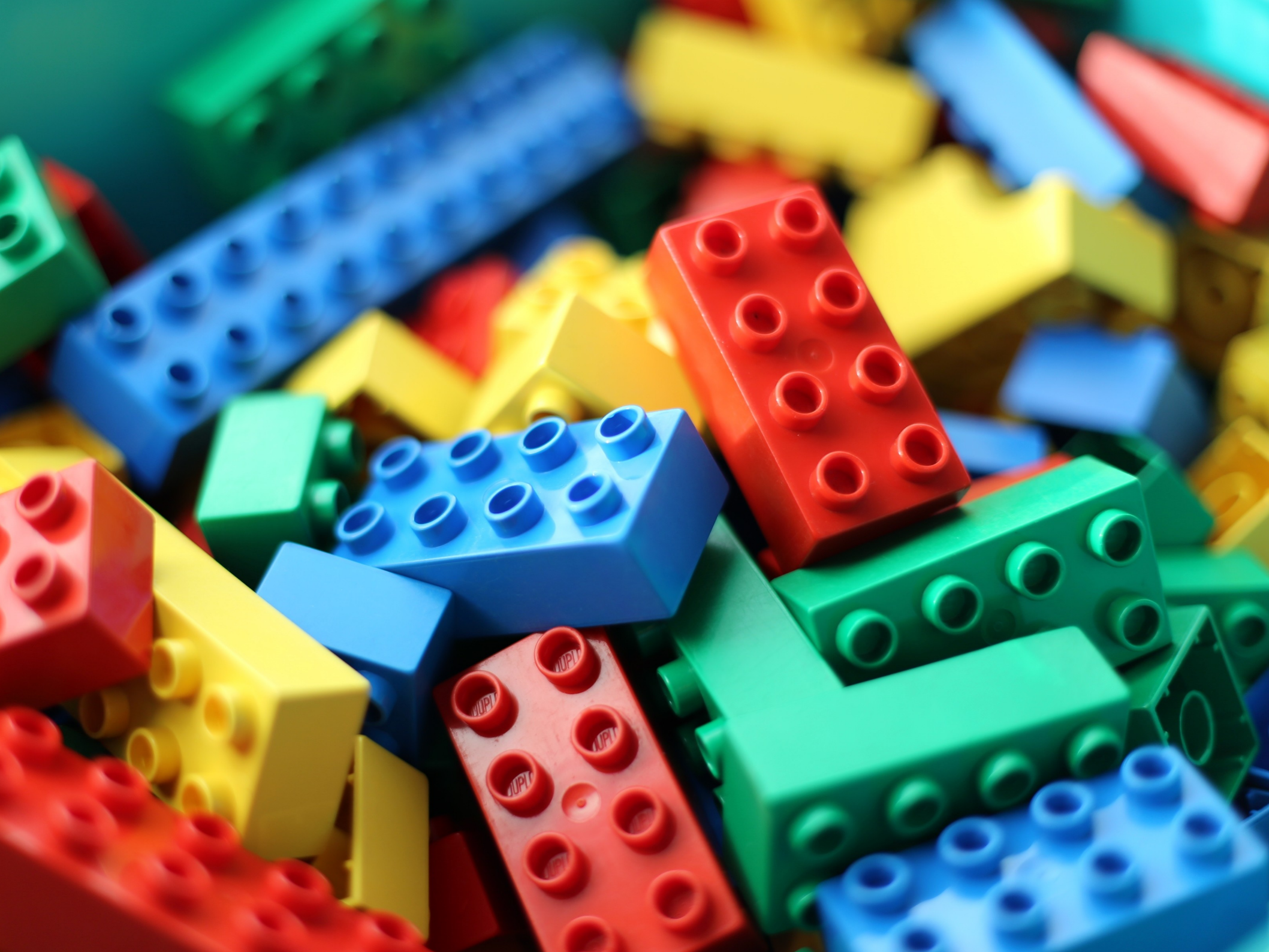
- Isolerer prosesser / apper
- Deler Kernel
- Deler ressurer
  - Men kan kvoteres!
- Isolerer prosesslister, brukerlister, filsystem, med mer
- Linux i bunn. Flere skall oppå.
  - Docker, rkt, lxc, systemd-nspawn

Docker?

- Resultatet av 1000 aper med hver sin skrivemaskin
- Gjorde containere populært!
- Dockerfile - oppskrift for å lage et image og hvordan man skal kjøre en container i en slags standardisert form
  - FROM debian
  - RUN apt-get foo
  - COPY foo /root/foo
  - EXPOSE 80
  - CMD /root/foo

Kubernetes?

- Kjører docker i bunn (eller rkt i moderne tid)
- System for å orkestrere kjøring av containere
- Du har X maskiner og Y apper i Z containere - Kubernetes gjør det mulig å kjøre det hele.



Komponenter

Alt er små, enkle „dingser“

- I hjertet av alt har man „etcd“, en key-value store med clustering.
- etcd er det eneste som faktisk har tradisjonelle cluster-krav: Det er her du må tenke på antall noder du har for å få quorum, osv.
- Stort sett dønn stabilt. Veldig sjelden du egentlig må jobbe direkte med det.
- Ikke egentlig en direkte del av Kubernetesprosjektet.

Control plane

- Det er ingen „master“ i Kubernetes. Uansett hva google sier. Masteren ligger i etcd.
- Du har et „control plane“, dette er tjenestene som gjør at Kubernetes funker.
- Alle tjenestene i control planet kan kjøre enten i kubernetes eller utenfor.
- I tillegg har man „kubelet“ som er komponenten som faktisk starter containere.

Node

- Alle noder har kun to komponenter som minimum
- kubelet er absolutt nødvendig - den starter og stopper ting
- kube-proxy er „ganske“ nødvendig - den gjør tjenestenettverk mulig (mer om det senere).

API

- kube-apiserver
- Alt går gjennom kube-apiserver
- kube-apiserver gjør ingenting annet enn å koordinere med etcd, samt sikre en del tilgang
- Uten kube-apiserver så har du ikke et cluster
- Pluggbart system for autentisering, men i utgangspunktet er SSL-sertifikater veldig populært
- Autentiserer mot etcd typisk med SSL-sertifikater - og er eneste komponent som gjør det

Tilgang? kubeconfig

- Klient-verktøy og server-komponenter bruker samme config-fil for tilgang: kubeconfig
- Lagret i ~/.kube/config normalt
- Eneste komponent som ikke bruker denne er kube-apiserver, siden det er denne som autentiserer alle andre komponenter
- (insert „rask titt“ her)

Kubelet

- Kubelet kjører på alle noder og har én jobb: Start og stopp ting.
- Kubelet henter informasjon fra to steder:
  - kube-apiserver
  - „manifestfiler“ på disk (eller en URL)
- Sistnevnte gjør det mulig å starte api-serveren inni kubernetes: Definer kube-apiserver i en manifest-fil så vil kubelet starte den selv, deretter koble til den.
- Kubelet eksponerer også statistikk

Litt mer om kubelet

- Kubelet er simpel, men det er en del „brytere“
- Sansynligvis (!) her man vil måtte skru litt for å eksponere for eksempel GPU‘er (hvis noen mot formodning måtte ønske det!)
- Kan snakke med et cluster med ssl-sertifikater. Det er en egen prosedyre for å bootstrappe dette: Kubeleten kan selv spørre om nye sertifikater.

Kube proxy

- kube-proxy er en liten tjeneste som kjører på alle noder
- Den setter opp tjenesteadresser lokalt
- Typisk med iptables, ipvs eller user-space.

Controller manager

- API‘et alene gjør ingenting
- controller-manager består av en „dullion“ små moduler som faktisk gjør ting
- Garbage collection, ssl-signering, deployments, etc
- Sjekk „kubectl get clusterrole --all-namespaces“

Scheduler

- kube-scheduler er ansvarlig for å fortelle kubelts, indirekte (via api‘et), hva de skal kjøre. Og det er alt.
- Sjelden eller aldri noe man må fikle med.

DNS

- kube-dns, eller coredns i nyere installasjoner, gjør det mulig å slå opp „tjeneste.navnerom.svc.cluster“
- Ofte ikke regnet som en del av control planet...

„misc“

- Mange støttetjenester faller ofte utenfor definisjonen av control planet:
  - heapster for agregert statistikk
  - fluentd for logging av app-logger
  - eventer for logging av eventer
  - osv
- Det eneste man _må_ ha er: kubelet, kube-apiserver, kube-scheduler og kube-controller-manager.

Nettverk

PODer

- En pod er en samling av en eller flere containere.
- En pod er den minste logiske enheten man jobber med i Kubernetes. Alle containere i en pod kjører på samme maskin og med samme nettverksoppsett.
- Containere innen samme pod kan snakke sammen ved å bruke localhost. F.eks. localhost:80.

Pod-nettverk

- Hver node har et pod-nettverk.
- Alle podder må kunne snakke sammen, på tvers av noder
- Det finnes flere måter å implementere dette på:
  - overlaynettverk
  - „vanlig“ nettverk (krever ruting)
- Det er ikke nødvendig at pod-nettverk er nåbare fra internett.

POD IP‘er

- POD ip‘er er flyktige (ephemeral). Hver gang en pod blir flyttet på får den en ny IP.
- Å bruke Pod IP‘er direkte er ikke vanlig eller ønskelig

Tjenester: Services

- Tjenester etableres ved å lage „service“-objekter
- En Service referer normalt til en logisk samling av pods (vi kommer tilbake til dette)
- En service kan enten være „headless“ og kun eksistere i DNS (tenk DNS round robin med alle Pod-ip‘er dynamisk oppdatert)
- Eller som en statisk service IP.
- Sistnevnte er der kube-proxy kommer inn i bildet.

Servicenett

- Service IP‘er trenger kun være tilgjengelig internt i clusteret
- kube-proxy oversetter til pod-ip‘er: du trenger ikke å kunne rute service-iper internt.
- Det hender det er ønskelig å gjøre service-nettet rutbart, men det er ikke et krav.
- Bør sette av mange adresser (/17 f.eks)

Trafikk fra internett?

- Finnes forskjellige metoder - selvsagt
- „Ingress“-objekter for lag 7 (http/https)
- „LoadBalancer“ for lag 2/3
- „Service“ med „NodePort“
- Rute servicenettet
-
- Uansett peker alt på en service. Det finnes ingen fasit.

Grunnprinsipper for bruk

Objekter objekter objekter

- Alt i Kubernetes er representert som objekter, typisk represnetert som YAML.
  - Støtter også JSON
- Bruker samme metode for å modifsere lagringsoppsettet som en container
- REST-basert
- Bruk „kubectl“ CLI-verktøyet.
- Operasjoner er stort sett „cascading“. Sletter du et objekt så slettes biproduktene også.
  - Unntak: Persistent lagring

Objektoppbygning

- apiVersion:
  - Definerer api-gruppe. Kan endres etterhvert som nye API‘er modnes. Typisk „v1“ „v1beta2“ osv.
- metadata:
  - Metadata om objektet, deriblant alltid et navn. Det meste annet er valgfritt. Type: creationTimestamp, generation, etc.
- spec:
  - Den spennende biten – hva gjør denne greia?
- status:
  - Nåværende tilstand (om noen)

Bruk referansedokumentajsonen for detaljer.

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.9/

(eller google det)

Hvorfor containere i Pod?

- Flere containere samarbeider om én oppgave
- Gjenbrukbare containere
- F.eks.: Standard nginx-container for å eksponere en webserver, og en fil-hente-container som populerer volumet nginx-containeren bruker.
- F.eks.: Leader Election „sidecar“ som kun velger en leder blant mange replicas. Hovedcontainer sjekker med sidecar-containeren om den er blitt leder.

Objekter – se på

- „kubectl get <objekttype> [objektnavn]“
- kubectl get deployment
  - Lister alle deployments i tekstbasert form
- kubectl get pod foo
  - Lister kun en navngitt pod
- kubectl get service -o yaml
  - Lister alle servicer med alle detaljer, i yaml-format

Objekter – se på mer?

- „kubectl get“ er fint for å se på kun en ting
- „kubectl describe“ er „context aware“ og er generelt nyttig for feilsøking/dekoding.
- „kubectl describe pod foo“
  - Lister f.eks. status på helsesjekker, event-loggen for pod‘en, med mer.

Objekter - modifisering

- „kubectl create -f <foo.yaml>“
- „kubectl replace -f <foo.yaml>“
  - Ikke-destruktiv – vil normalt gjøre f.eks. Rolling update.
- „kubectl replace -f <foo.yaml> --force“
  - Potensielt destruktiv – sletter først det gamle objektet og lager det på nytt.
  - Nødvendig om man f.eks. endrer port på en tjeneste

Objekter - modifisering

- Kan også kjøre „kubectl edit <objekttype> [navn]“
  - Åpner editor og gjør en „kubectl replace“ bak kulissene.
- Andre alternativer: „kubectl apply“ og „kubectl patch“.
- Finnes også en rekke short-hands:
  - kubectl set image deployment/frontend nginx=nginx:1.9.1
  - kubectl expose deployment foo
  - kubectl scale deployment foo --replicas=5

Navnerom?

- Navnerom isolerer prosjekter eller under-prosjekter.
- Et navnerom er nettopp det, men har også rettigheter og ressurskvoter knyttet opp mot seg
- Tanken er å begrense uhell, ikke bruk
  - Unngå at du tar ned naboens prosjekt fordi du blingser med en null og deployer 60 i stedet for 6 instanser
  - Trenger du mer? Spør.
  - I dag kan namespaces lages via http://kubernetes/

Kubectlversjon?

- Du kan fint bruke en gammel kubectl.
- Hovedforskjellen er at den vil mangle „short hands“ for nyere konstruksjoner.
- De kan fortsatt jobbes på med standard create/edit/ replace.
- Typisk forbedringer: Rikere output. Flere kommandolinjesnarveier som lager yaml for deg.

Tilgang til et navnerom?

- En admin i navnerommet fikser.
  - „kubectl create rolebinding --namespace <foo> --user=n07200 --clusterrole=edit“
- AD-baserte grupper er på vei, og vi vil nok snart se at man kan bruke n-bruker+passord direkte i stedet for klientsertifikater.
- Du vil ikke miste tilgang hvis/når dette byttet skjer. Gammel tilgang vil fortsette å fungere inntil videre uansett.

Demo: Oppsett av klient.

Deployment

- En deployment er det du normalt jobber med
- Definerer en template for å lage en Pod
- Definerer hvor mange replicas av denne Pod‘en som ønskes
- Styrer hvordan oppdateringer skjer
  - Typisk Rolling Update.
- Sletter du Deploymenten slettes normalt Podene også.
- Sletter du Pod‘ene først så vil Deploymenten bare lage dem på nytt.

Demo: Teste eksempel-prosjektet

Lagring

Lagring

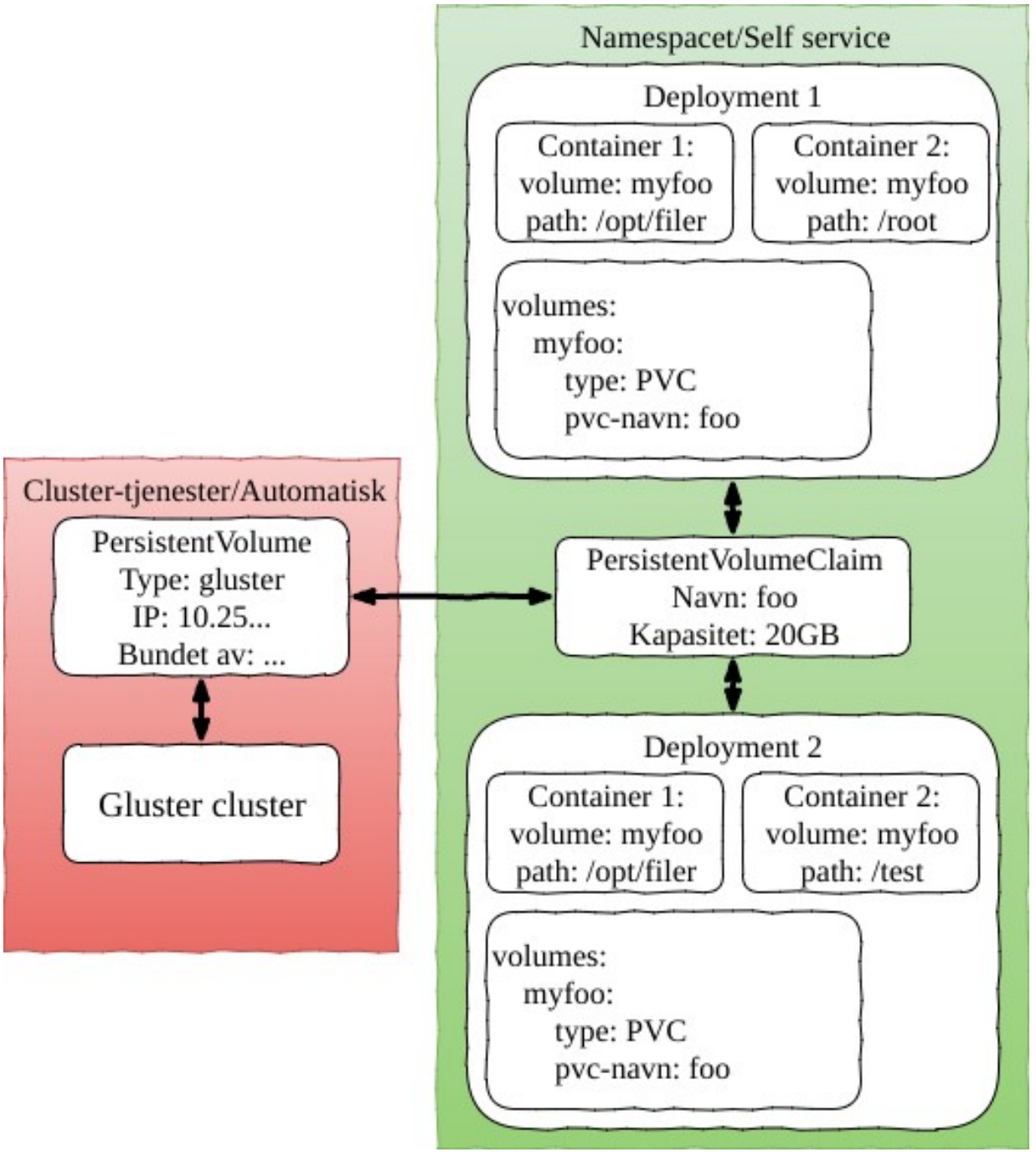
- Du kan selv sette opp begge deler – og det er isolert.
- Trenger du mer plass? Du kan resize volumene dine „on the fly“ (kun øke, ikke redusere plass)
- Bruk ConfigMap til konfigurasjon, eventuelt Secrets som fungerer likt, men er kun synlig for brukere med rettigheter i navnerommet

Fra begynnelsen: Volume

- Kubernetes lar deg blande miljøvariabler og lagring sammen i en stor mølje
- En container referer til et volum og en path det skal monteres.
- Dette volumet kan tilfredsstilles av en rekke forskjellige mekanismer:
  - NFS, Gluster (via PersistentVolumeClaim), ConfigMap, Secret, metadata om pod‘en, etc
- Mange av de samme mekanismene er tilgjengelig for å populere miljøvariabler: ConfigMap, Secrets, metadata om poden.

Scenario: „Vanlig lagring!“

- Gluster.
- Dette gjøres ved at du referer til et „Persistent Volume Claim“ (PVC).
- Du må også lage PVC‘et.
- Clusteret vil da automatisk provisjonere opp lagring og opprette et Persistent Volume som er „ditt“.
- Så lenge PVC‘et eksisterer så er lagringen din.
- Du kan referere til samme PVC fra flere deployments.


Namespacet/Self service
Deployment 1
Container 1:
volume: myfoo
path: /opt/filer
Container 2:
volume: myfoo
path: /root
volumes:
myfoo:
type: PVC
pvc-navn: foo
Cluster-tjenester/Automatisk
PersistentVolume
Type: gluster
IP: 10.25...
Bundet av: ...
PersistentVolumeClaim
Navn: foo
Kapasitet: 20GB
Gluster cluster
Deployment 2
Container 1:
volume: myfoo
path: /opt/filer
Container 2:
volume: myfoo
path: /test
volumes:
myfoo:
type: PVC
pvc-navn: foo

Arbeidsmåte

- PVC‘er lages typisk en gang, og endres kun om man trenger mer plass.
- Det eneste de har av informasjon er et navn, bruksmåte (Read only/Read-write one/Read-write many) og kapasitet.

ConfigMap og Secrets

- ConfigMaps og Secrets er 98% likt
  - (Svæt vitenskapelig estimert prosent)
- Eneste praktiske forskjeller:
  - ConfigMap kan lagre litt mer data
  - Kun medlemmer av et namespace kan se innholdet i en secret
- Tenk passord versus configfiler.
- Brukes identisk. Snakker jeg om ConfigMaps kan du anta det samme gjelder secrets.

ConfigMap: Lagre config

- Key-value store, kan lagre hele filer eller enkelt-variabler
- Kan mappes inn i en container enten som filer eller miljøvariabler.
- Namespacet: Har dere et prod-namespace og et dev-namespace? Bruk et ConfigMap i hvert for å samle config.

Deployment versus StatefulSet

- I en Deployment vil alle replicas ha tilgang til nøyaktig samme lagring. PVC‘en defineres utenfor Deploymenten.
- I et StatefulSet vil hver replica ha unik lagring – PVC‘en defineres som en template.
- En deployment med 10 replicas bruker like mye disk som en med 1 replica.
- Et StatefulSet med 10 replica bruker 10 ganger så mye disk.
- Sistnevnte vil lage nye PVC‘er om den skaleres opp.

Bruk Deployment når...

- Applikasjonen selv kan håndtere leader election/låsing av data
- Når alle instanser av applikasjonen skal ha tilgang til samme filområde.
- Eksempler:
  - Webserver med statiske filer

Bruk StatefulSet når...

- Hver instans trenger unik, isolert, lagring
- Du trenger at applikasjonen starter i fast rekkefølge (Start først 1, så 2, så 3 – hver gang)
- Eksempler:
  - Databaser med replikering (finnes massevis av eksempler, f.eks. MySQL)
  - Applikasjoner som trenger lagring til arbeidsområder.
  - RabbitMQ?

Nettverk!

Pod-nettverk

- Alle Poder har en unik IP.
- Disse er nåbare fra NRK – DMZ har nok firewalling.
- Alle Poder i et cluster kan snakke med alle andre i clusteret.
- Disse adressene er flyktige – hver gang en Pod flyttes/lages så får den en ny IP.
- Selve IP‘en er en funksjon av hvilken Kubernetes-node den er på.
  - Pod‘en med IP 10.25.102.25 befinner seg på noden med ip 10.25.0.102 (malxkube02)

Service

- For å nå en tjeneste bruker man servicer. Disse er separate fra Deployment/StatefulSet, men referer til dem.
- Servicer kommer i flere varianter. Vi bruker to:
  - ClusterIP (default)
  - Round Robin DNS (sett clusterip til „none“)
- Alle servicer har DNS-navn!
  - <servicenavn>.<namespace>.svc.<sone>.nrk.cloud
  - foobar.n07200-test.svc.int.nrk.cloud

Servicer: Round Robin

- Den enkleste servicen er kun basert på DNS.
- Sett ClusterIP til „none“.
- Slår du opp navnet i DNS får du IP‘en til alle Podene som matcher akkurat nå.
- Har alle de fordeler og ulemper som round robin DNS har.

Service: ClusterIP

- Allokerer en statisk IP til tjenesten i 10.25.4.0/22-nettet (eller 10.26.4.0/22-nettet for DMZ)
- Denne IP‘en peker alltid på riktig service internt i clusteret.

Ingress: Lag 7, HA

- I tillegg til Servicer kan man ingresse trafikk via,… err… Ingress-objekter.
- Implementert ved hjelp av BigIP som tar seg av fail-over og kommuniserer med et redundant cluster av nginx ingress-controllere.
- Ikke ulikt nixy.
- Du må selv lage Ingress-objektet.
- Ingen namespacing!

Mer om Ingress

- Ingress går ikke ned om en node går ned utover det åpenbare (er halve requestet sendt, osv…)
- Ingress har ingen kapasitetsbegrensinger
- Vi har satt opp *.kubeint.nrk.no og *.kubedmz.nrk.no som wildcards som kan brukes av ingresser. Med SSL-terminering.
- Andre domener kan også settes opp etter behov/ønske.

Cluster-ekstern trafikk

- For å rute 10.25.4.0/22 inn i clusteret er det rutet via 10.25.0.10 – en enkelt IP, allokert til en enkelt maskin.
- Denne IP‘en flytter seg automatisk om maskinen den er på går ned.
- Uansett betyr det at ALL trafikk til service ip‘er går via samme maskin (begrenset båndbredde) og er sårbare for nedetid på 5-20 sekunder om maskinen går ned.
- Internt i clusteret er dette ikke tilfelle – service IP‘en blir „resolvet“ før den forlater noden.

Hvordan velge?

- Du må uansett ha en service i bunn – Ingresser peker på servicer (husk: Internt i clusteret er det HA).
- Ingress støtter kun HTTP og HTTPS.
- Er trafikken kun intern i clusteret: bruk service.
- Takler tjenesten nedetid på 5-25 sekunder ved patching/reboot og har ikke store mengder data som skal overføres? En service holder sansynligvis.
- Trafikk fra publikum? Bruk Ingress!

I DMZ?

- I DMZ vil uansett ingresser være å foretrekke – det er ikke gitt at du har brannmuråpninger.

Litt mer avansert: Helsesjekker, ressurser

Helsesjekker

- Finnes i flere varianter, deriblant HTTP/HTTPS-prober, TCP og kjøring av arbitrære kommandoer.
- „livenessProbe“: Lever denne greia? Hvis ikke: Drep den og start på nytt.
- „readynessProbe“: Er denne klar til å motta trafikk?
- Ingen er påkrevd. Readynessprober kan brukes hvis en applikasjon bruker lang tid på å starte f.eks.

Ressurser

- Vi har satt kvoter for CPU, minne, antall pods og persistent storage – per namespace.
- Dette kan økes etter ønske/behov.
- Mer kan komme (f.eks. Begrensing i ephemeral storage)
- Kan studeres ved dashboardet eller „kubectl describe resourcequota“
- Sysadmins kan endre det (med „kubectl edit resourcequota – superavansert).

I praksis

- Hver container spesifiserer ressurser på to måter: „request“, som er det containeren MÅ ha tilgjengelig for å fungere, og „limit“ som er øvre grense.
- Enheter er byte og cpu‘er.
- Eksempel: request 0.2 CPU, 256MB RAM. Limit: 2CPU, 1024MB.
  - Pod‘en er garantert 0.2CPU‘er og 256MB ram, men kan fritt bruke 2 CPU-kjerner. Bruker den mer enn 1024MB OOM‘er den.

Defaultverdier og grenser

- Hvert namespace har også en LimitRange
- LimitRange spesifiserer tillate verdier og default-verdier for ressurser til en Pod
- Eksempel:
  - request: minimum 0.05 CPU, default 0.2, maks: 2
  - limit: minimum 0.5 CPU, default 2, maks: 10

Best practice

- CPU: Gjett villt. Request kan være det applikasjonen bruker normalt, men limit bør være minst 1 eller 2 så ting går rundt når det trengs.
- Minne: Det gir ikke veldig mye mening å skille stort på Request og Limit. Bruker appen din mellom 1 og 20 GB minne så sett request og limit til 20+GB
- Ellers vil den drepes.
- Unntak: applikasjoner som tidvis gjør bulkjobber og kan recovere om de blir drept fordi noden er tom for minne.

Hvorfor bry seg?

- Det påvirker driftsstabilitet.
- Særlig limit på CPU er viktig ved burst-trafikk.
- Setter du for lav limit på minne vil applikasjonen OOM‘e unødvendig.
- Setter du for lav request på minne vil applikasjonen OOM‘e tidligere enn andre på samme node om noden går tom for minne.

Hva med DSx?

- Hver node er lablet med en zone.
- Kubernetes prøver å spre poder innen samme deployment:
  - Jevnt over forskjellige soner
  - Jevnt over forskjellige noder innen samme sone
- Dette er INGEN garanti
  - Fordrer kapasitet osv
- Kan overstyres med „affinity“

PodDisruptionBudget med mer

- Når noder booter blir de først drainet for applikasjoner – det krever at applikasjonen din kan kjøre på mer enn et sted samtidig.
- Har du mange replicas kan du med PodDistruptionBudgets spesifisere hvor mange som kan være „disrupted“ samtidig: skal tre pod‘er flyttes kan du si at clusteret kun får lov til å flytte en om gangen.
- Bør ikke være nødvendig.

Generelle råd

- KISS
- Forvent at ting blir startet på nytt av og til (typisk ved patching)
- Har applikasjonen spesielle krav til ytelse så er ressurser viktig, ellers er defaultene sansynligvis greie nok (håper vi)
- Prøv å støtt kjøring av flere instanser av appen samtidig.
- Vi trenger mer „brukerforum“ for å finne best practices.

Feilsøking og utforsiking

Generelt: Eksperimenter.

Det er vanskelig å gjøre skade på andre – og det er forholdsvis lett å rydde.

Feilsøking 101

- I starten er dashboardene fine!
- http://int.nrk.cloud
- Tips: Velg ditt eget namespace og let etter stygg, rød tekst!
- På sikt går det ofte like fort å gjøre „kubectl describe ...“, men teksten blir jo ikke rød.

Kjører Pod‘en?

- Sjekk om den i det heletatt kjører. „kubectl get“…
- Normalt vil „describe“ forklare hvorfor den ikke gjør det.
- Har det skjedd noe „funky“? Du kan alltids slette Pod‘en – Deploymenten vil lage den på nytt!
- Vanlig feilkilder om en Pod ikke kjører:
  - Feil image (har du husket å pushe?)
  - Feil i volum/mounts. (Har du lagd PVC‘et?)

Pod‘en kjører, hva nå?

- „kubectl logs ...“ - finnes også i kibana.
- „kubectl exec -ti minpod-124121 /bin/bash“ !!!
  - Anbefales på det sterkeste å leke med dette.
  - Du kan teste ting rett i Pod‘en din – uten å spørre Linuxadmins.
- Generelt, sjekk „kubectl –help“
- „kubectl cp“ for å kopiere filer f.eks.
- „kubectl port-forward“ om du trenger å port-forwarde en lokal port inn i clusteret (nyttig i DMZ).

Eventer

- „kubectl get events“ - Eventer av typen „Deployment skalert opp“, „Pod restartet“, osv.

Følg en Deployment til bunn

- „kubectl describe deployment foo“
- Se etter „ReplicaSet“ - dette er instansen av deploymenten
- „kubectl describe replicaset ….“
- Når du endrer en deployment så lages et nytt ReplicaSet som matcher de nye endringene, dette skaleres gradvis opp mens det gamle skaleres ned.

Automatisk bygg?

Generelt

- Hold ting i git – begrens bruken av „kubectl edit ...“. (Det finnes gode unntak)
- Bruker du „kubectl replace „ uten „--force“ er du stort sett trygg.
- Du kan referere til hele kataloger.

Jenkinseksempel

- Standard jenkins på malxmetadev01(?) har en slave (malxorigokubectl01) som kun har en funksjon: kjøre kubectl.
- Fungerer egentlig fint nok.
- Finnes MANGE alternativer, deriblant å kjøre hele Jenkins i Kubernetes slik at den selv spinner opp sine egne slaver.

Teamcity?

- Også mulig – er satt opp som byggeslave for MAM-prosjektet/Tedial (usikker på om den er i bruk).
- Deployment er greit nok.

ServiceAccounts?

- Alle applikasjoner som kjører i Kubernetes er knyttet opp mot en Service Account. Normalt „default“-accounten i namespacet den kjører.
- De fleste kubernetes-verktøy (inkludert kubectl) bruker service accounts automatisk om de er mountet opp.
- Gjør det lett å gi applikasjoner i clusteret tilgang til å endre eller se på clusteret. F.eks. Jenkins.

Annet?

- Batch-jobb? „Job“
- CronJob.
- HorizontalPodAutoscaler – start ekstra replicas om X% CPU er brukt.

- Jevnt over er det alt for mye man kan gjøre.

???