Hovedoppgave

NTNU
Norges teknisk-naturvitenskapelige universitet

Adis Pinjic og Anders Kvanvig

Containeroppsett i Windows Server 2019

Hovedoppgave i Informatikk, drift av datasystemer

Mai 2019

NTNU
Kunnskap for en bedre verden

Adis Pinjic og Anders Kvanvig

Containeroppsett i Windows Server 2019

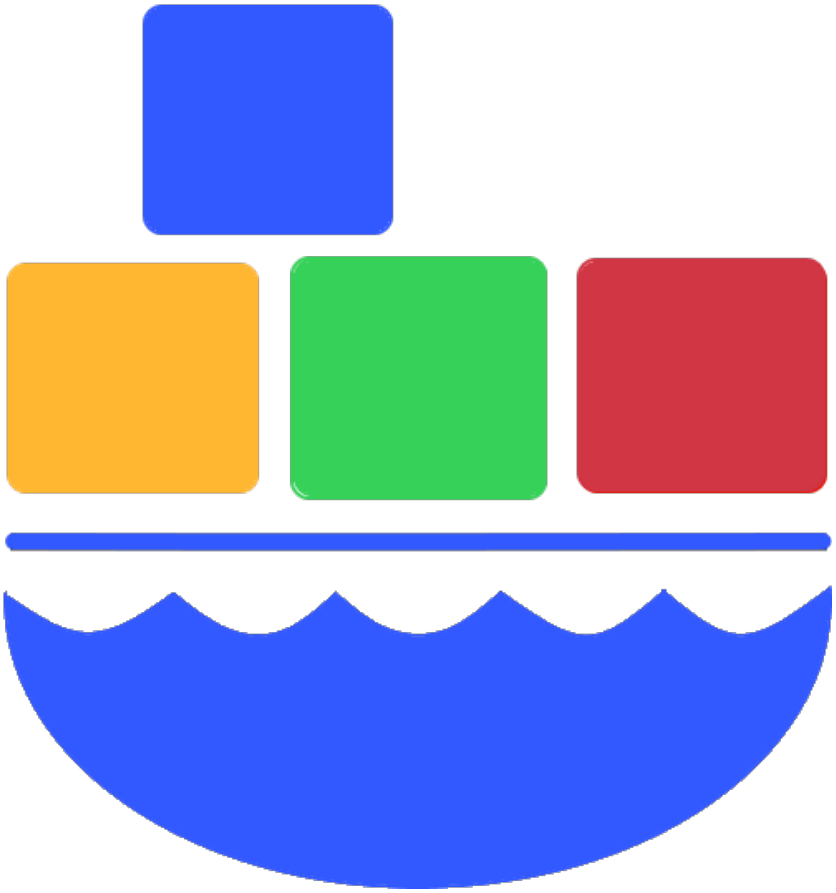
Hovedoppgave i Informatikk, drift av datasystemer
Mai 2019

Norges teknisk-naturvitenskapelige universitet

NTNU
Kunnskap for en bedre verden

Innhold

Containeroppsett i Windows Server 2019



Forstudierapport

Anders Kvanvig
andekva@stud.ntnu.no

Adis Pinjic
adisp@stud.ntnu.no

Revisjonshistorie

Versjon	Dato	Forfatter(e)	Beskrivelse
1.0	07.02.19	Adis Pinjic, Anders Kvanvig	Første utkast fra prosjektgruppen.
1.1	14.02.19	Adis Pinjic, Anders Kvanvig	Oppdatert rapporten basert på tilbakemeldinger fra tredje veiledermøte. Endringer på interessentanalyse og informasjonsbehov. Lagt til Revisjonshistorie og figurliste.
1.2	12.04.19	Adis Pinjic, Anders Kvanvig	Gjennomlesing og mindre feilretting + forside
2.0	26.04.19	Adis Pinjic, Anders Kvanvig	Lagt rapport inn i LaTeX

Innhold

Figurer

1 Introduksjon - Hensikten med dokumentet

I denne forstudierapporten vil et forskningsarbeid utføres for å utforske mulighetene for å bruke containerløsninger i Windows Server 2019 sammen med Kubernetes. Løsningene vil rettes mot bedrifter med større driftsmiljø, således baseres oppgaven på å:

- Øke driftseffektivitet på stor skala
- Gjøre rede for administrasjon av containertjenester
- Demonstrere et mulig oppsett for Kubernetes-miljø med Windows noder

Forstudiet dokumenterer følgende:

Bakgrunn for prosjektet – Beskriver hvorfor behovet for oppgaven oppstod hos oppdragsgiveren.

Prosjektmål - Hvilke mål oppdragsgiveren og bachelorgruppen selv har satt for oppgaven. Punktet vil beskrive omfanget av arbeidet og hvilke hovedmål og milepæler som er satt for oppgaven.

Interessenter og rammebetingelser – Tar for seg hvem som har interesse av oppgaven og sluttresultatet som oppnås. I tillegg vil bestemte rammebetingelser presenteres, samt faktorer som eiendomsrett, involvering og bedømmelse underveis.

Kritiske suksessfaktorer – Bestemte mål bachelorgruppen ønsker å oppnå for at arbeidet kan anses som vellykket. Videre listes hva som skal leveres av dokumentasjon.

Teori og analyse – I dette punktet presenteres generell teori om containere for å gi interessenter og lesere et innblikk i hvordan teknologien fungerer. I videre analyse diskuteres hvilke bruksområder containere kan ha i bedriftssammenheng.

Risikoanalyse – Beskriver hvilke faktorer som kan påvirke arbeidet og hvilke konsekvenser de kan medføre. Bachelorgruppen vil i tillegg foreslå tiltak som kan minke sannsynligheten for at konsekvensene inntreffer.

Retningslinjer og standarder – Krav for dokumentasjonen som skal leveres, samt hvem som skal gjennomføre kvalitetskontroll.

Prosjektorganisering – Hvordan arbeidet gjennomføres, hvilke deltakere som er med og deres roller.

Anbefaling om videre arbeid – Dette punktet vil oppsummere resultatene av forstudierapporten og gi en anbefaling for videre arbeid i bacheloroppgaven.

2 Bakgrunn for prosjektet

Containere, mikrotjenester og Docker har alle vært populære trender innen IT-infrastruktur de siste årene. Teknologien kan anses som et naturlig tredje steg etter den første fasen med fysiske servere og den andre fasen med virtualiserte servere. Mengden med aktører som migrerer tjenestene sine til skyplatformer øker for hvert år, og alle ønsker å gjøre skyen sin så effektiv som overhodet mulig. Enorme virksomheter som Netflix, Amazon, LinkedIn, Twitter og Spotify har allerede gått over til komplette oppsett som baseres på containerteknologi, noe som fort vekker oppmerksomheten til resten av næringslivet.

Containerløsninger har lenge eksistert på Linux, og plattformen har per i dag de mest optimaliserte oppsettene for containerdrifting. Minuset med containere på Linux er at de ikke kan kjøre Windows-containere, noe som har gitt Microsoft en mulighet for å implementere sine egne løsninger på markedet. Containere ble først introdusert ved Windows Server 2016 sammen med mer lettvektige versjoner av operativsystemet. Nå i 2019 har selskapet publisert samarbeidsverket sitt med den største containerutvikleren, Docker, i form av en oppdatering av Windows Nano Server. Denne utgaven av Windows begynner å nærme seg nivået av lettvektighet en har sett i Linux, noe som viser Microsoft sin intensjon om å satse videre innen feltet.

Denne nye containersatsingen i Windows har vekket interesse hos IT-virksomheten Basefarm, som har gitt prosjektdeltakerne muligheten til å utføre et forskningsarbeid om dens bruksmuligheter. Basefarm AS er en europeisk driftsleverandør som i 2018 ble kjøpt av skytjenestekonsernet i telekomgiganten Orange. Virksomheten tilbyr i dag tjenester innen sky, datasenter, sikkerhet og big data.

3 Prosjektmål

Prosjektmålene for bacheloroppgaven deles inn i 3 deler:

Effektmål - Beskriver hvilken ønsket effekt resultatet av prosjektet vil ha for fremtiden.

Resultatmål - Beskriver hva som skal oppnås knyttet til prosjektets resultat.

Prosessmål - Beskriver mål for prosjektdeltakernes egenutvikling gjennom prosjektarbeidet.

3.1 Effektmål

- Kunnskapen som oppnås om containerløsninger vil kunne bidra til at interessentene for oppgaven kan bedømme Windows-containere til sine bruksmål.

3.2 Resultatmål

- Tjenermaskiner satt opp med Windows Server 2019
- Containertjenester i Docker
- Administrasjon av containere i Kubernetes
- Fungerende lastbalansering mellom tjenerne
- Automatisk skalering av tjenester
- Demonstrert oppsett for sikring og overvåkning av containere
- Dokumentasjon og rammebetingede rapporter

3.3 Prosessmål

- Få innsikt i bruk og oppsett av containerløsninger på Windows.
- Erfaring i hvordan en tar for seg ny teknologi
  - Kartlegge nyttige bruksområder
  - Implementasjon av tjenester
  - Formidle kunnskapen videre

3.4 Prosjektets omfang og milepæler

I prosjektet skal gruppen ta i bruk:

- Windows Server 2019
- Nano Server og Windows Server Core
- Docker
- Kubernetes

I prosjektet kommer deltakerne i tillegg til å da i bruk diverse tjenester, eksempelvis en web-applikasjon i testmiljøet.

En plan for prosjektets hovedoppgaver kan finnes i vedlagt Gantt-diagram. Filen er opprettet i MS Project 2016 og blir lastet opp på bachelorgruppens Sharepoint-side som deles med oppgavens interessenter.

4 Interessenter og rammebetingelser

4.1 Interessentanalyse

Prosjektdeltakerne:

Bachelorgruppen ønsker å opparbeide kompetanse innen containerteknologi gjennom arbeidet med oppgaven. Arbeidet gir deltakerne muligheten til å få uvurderlig erfaring innen moderne IT-drift, samt hvordan en skal gå frem for å undersøke og ta i bruk innovasjoner innenfor fagfeltet.

Veiledere:

- Stein Meisingseth

  Meisingseth vil i sammenheng med bacheloroppgaven være prosjektgruppens kvalitetskontrollør og være en del av den karaktersettende vurderingen av arbeidet for NTNU. Hans tidligere erfaringer i næringslivet og veiledning i tidligere bacheloroppgaver vil være verdifull for prosjektdeltakerne underveis. Som interessent ønsker Meisingseth å få innsikt i Windows Server 2019 og containerløsninger. I tillegg vil han naturligvis som foreleser at hans studenter har mulighet til å oppnå den tiltenkte kunnskapen og ferdighetene de skal ha etter bacheloroppgaven er fullført.

- Patrik Storm Olsen

  Olsen er gruppens veileder fra oppdragsgiveren Basefarm AS. Gjennom arbeidsprosessen vil hans innsyn i bedriftens behov og interesser forme det praktiske oppsettet, samt hvilke aspekt med oppgaven gruppen skal legge mest fokus på. Som interessent er ønsket hans å få en videre innsikt i containerteknologi i Windows og ha et gjensidig suksessfullt samarbeid med studentene han er oppdragsgiver for. Olsen og Basefarm stiller i tillegg med maskinvare for testmiljø etter kartlagt behov.

- Henrik Johnsen

  Johnsen vil i samarbeid med veileder Meisingseth gi tilbakemelding på det praktiske arbeidet som blir utført av prosjektdeltakerne. Johnsen ønsker som interessent å få bedre innsikt i Windows Server 2019 og containerløsninger på plattformen.

Teknisk støtte:

- Sindre Stubberød

  Stubberød er en deltaker fra Basefarm AS og tilbyr teknisk veiledning for prosjektdeltakerne.

Basefarm AS:

Som oppdragsgiver ønsker bedriften å få innsikt i containere med Windows Nano Server gjennom prosjektdeltakernes arbeid. Bedriften vil i tillegg stille med maskinvare for det praktiske oppsettet.

NTNU:

Kommer til å publisere forsiden av oppgaven til offentligheten. Oppgaven vil være tilgjengelig for ansatte i fagenheten og elektronisk tilgjengelig på NTNU Open om ikke oppgavestiller setter begrensninger for dette. Et positivt samarbeid med IT-næringslivet vil også fremme studenter fra bachelorstudiet som verdige kandidater i prosjektsammenheng.

4.2 Rammebetingelser

- Prosjektstart: 7. januar 2019
- Frist for levering av prosjekt: 20. mai
- Beregnet tidsforbruk på oppgaven er ca. 500 timer per person
- Presentasjon av prosjekt vil foregå den 23. mai

4.3 Eiendomsrett

Eierskapet av oppgaven ligger hos prosjektdeltakerne, men arbeidet forbeholdes fritt tilgjengelig for både Basefarm AS og NTNU for deres behov.

5 Kritiske suksesskriterier

Kritiske suksessfaktorer er forutsetninger som skal oppfylles for at resultatet av prosjektet kan vurderes som vellykket.

5.1 Suksessfaktorer

Prosjektgruppen skal:

- Finne en komplett containerløsning for en større bedrift. Løsningen bør kunne sammenlignes med en tiltenk løsning som kunne vært i dag for å demonstrere effektiviteten av oppsettet.
- Få god innsikt i bruk av containere på Windows, i tillegg til at kunnskapen må kunne kommuniseres effektivt gjennom de rapportene som skrives. Leseren skal kunne få en overordnet innsikt i containerløsninger forutsett at leseren har forkunnskaper innen virtualisering og serverdrifting.

5.2 Informasjonsbehov

Følgende informasjon skal publiseres på nett som veiledere har tilgang til:

- Timelister
- Ukesrapporter
- Forstudierapport
- Designrapport/systemkravdokument
- Driftsrapport
- Sluttrapport
- Presentasjon
- Gantt-diagram

6 Teori og Analyse

6.1 Containere

6.1.1 Introduksjon

Conteinerteknologien bygger på det samme grunnlaget IT-utviklere hadde ved dannelsen av virtuelle maskiner. Det som ble oppnådd med VM-er var virtualisering av maskinvare. Med dette kunne flere tjenere og applikasjoner kjøre på samme maskinvare i stedet for på forskjellige fysiske maskiner. På den samme måten oppnår containere virtualisering av operativsystemet. I stedet for at hver VM må kjøre sitt eget fullverdige operativsystem, så vil containere bruke systemressursene fra en hostmaskin som den kjøres fra.

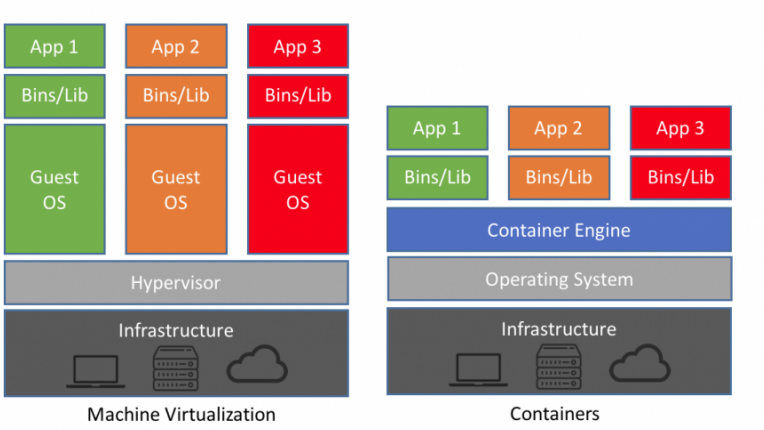
Conteinerløsninger er ikke en ny innovasjon, konseptet stammer nemlig fra tusenårsskiftet. Den tidligste introduksjonen av OS-virtualisering kom fra Linux "VServer"(Osnat, 2018) som partisjonerte ressursene på en hostmaskin. Grunnen til at containere ikke har blitt en trend før de siste årene stammer fra dagens muligheter til å effektivisere ressurser. Måten IT-næringslivet forsøker å oppnå dette er ved å være mest mulig Cloud Native og strukturere tjenestene sine som Microservices. Det er nettopp i dette rammeverket at containere kan eskalere fortjenesten til IT-bedrifter.

6.1.2 En mer effektiv sky

Det er mange grunner til at skytjenester har blitt sentrale innen IT-infrastruktur de siste årene. Ved å ha tjenester opp i skyen får virksomheter økt fleksibilitet, rimeligere og raskere backup, samt muligheten til å fjernstyre systemene sine. En slik effektivisering resulterer i lavere kostnader for driften av IT-tjenester, som igjen kan føre til økt fortjeneste på bakgrunn av nye konkurranseforde-ler. I lys av dette har mange IT-bedrifter hatt et klart mål om å bli "cloud native". Uttrykket går ikke ut på å bare flytte ut alle tjenestene sine til sky-en; en kan heller se på det som en metodikk for å stramme opp skytjenestene sine til å være så effektive som overhodet mulig. I dag defineres "cloud native computing" av CNCF (Cloud Native Computer Foundation) som løsninger basert på open source programvare i tillegg til tre sentrale krav(Janssen, 2018):

1. **Containerbasering** - hver del av en stack er pakket inn i sin egen container.

2. **Microservice-arkitektur** - alle applikasjoner og tjenester segmenteres til mikrotjenester. På denne måten kan bare de tjenestene som har behov for mer ressurser få de tildelt.
3. **Dynamisk orkestrering** - containerne blir aktivt administrert til å optimalisere ressursbruk.



Figur 1: Maskinvarevirtualisering sammenlignet med os-virtualiserings.
Kilde: (Chamberlain, 2018)

Figuren over beskriver godt hvordan en container fungerer i praksis. Containere sitter oppå en fysisk server og operativsystemet dets. Hver container deler på operativsystemet til hostmaskinen de sitter på, og rulles ut med alt den trenger for å operere. Dette medfører at om en har utviklet en container til å kjøre for eksempel en IIS-server, så vil containeren kunne kjøres på hvilken som helst maskin med en Container Engine (figur 1). Containere er i tillegg eksepsjonelt små i størrelse, og krever mindre administrativ overhead grunnet det delte operativsystemet. På grunn av dette anses containere til å være et lettvektig alternativ når det gjelder utvikling og kjøring av applikasjoner og tjenester i forhold til virtuelle maskiner.

Selv om containere har enormt mange bruksfordeler, så er det likevel ikke en fullverdig erstatter for virtuelle maskiner per i dag. Det finnes flere problemstillinger der containere ikke er en ideell løsning i dag:

- Applikasjoner kjører naturligvis ikke like raskt inn i en container som det ville gjort på en maskin kjørende direkte på operativsystemet. Har en tjeneste et spesielt behov for ytelse vil kanskje ikke en container lønne seg

- Containerløsninger er en ny teknologi som trolig kommer til å gjennomgå større endringer etter hvert som løsningene modnes
- Virtuelle maskiner er isolerte med sitt eget operativsystem og kjerne. Om en klarer å utnytte sikkerhetsfeil i en container kan det potensielt gå ut over hostmaskinen og påvirke alle containerne
- Om samme applikasjon eller tjeneste skal i tillegg kjøres på flere forskjellige operativsystem egner kanskje ikke containere seg siden de er OS-spesifikke

Microservices

Når en skulle utvikle en webapplikasjon for 5 år siden var det helt klart vanlig å baseres på monolittisk arkitektur. Med andre ord utviklet man gjerne hele applikasjonen i et enkelt repository(kodebase) som består av en database, en klientapplikasjon og en serverapplikasjon. Den store overgangen til skalerbare og skybaserte applikasjoner har vist seg å være lite effektiv i samarbeid med den monolittiske modellen. Tjenester og applikasjoner ble trøblete å skalere, utrulling tok lang tid og utviklingen ble ofte rammet av feil.

Microservices, eller mikrotjenester, baseres på å bryte ned de store monolittiske applikasjonene til en samling av mindre selvstendige tjenester. Eksempelvis kan en nettbutikk-stack brytes ned til fire komponenter:

- Front-end brukergrensesnitt
- Brukerdatabase
- Handlekurv
- Database for varer og tjenester

Hver del av oppsettet vil ha sin egen kodebase, noe som resulterer i at kommunikasjon mellom partene går gjennom API (Application Programming Interface). Ved å ikke integrere de ulike komponentene med avhengighet ovenfor hverandre så unngår man ulempene som gjør monolittisk arkitektur vanskelig i skyen. API-spørringer vil resultere i at utviklere lettere kan gjøre tjenester tilgjengelig for hverandre internt, i tillegg til at utvikling kan skje parallelt. Det er mange fordeler en kan oppnå ved å gjøre som mange andre og gå over til mikrotjeneste-arkitektur, spesielt trekkes det frem visse hovedgrunner(Kelly, 2017):

1. Utvikling skjer i mindre deler

Team kan jobbe parallelt på sin tjeneste uten å fokusere på de andre delene av applikasjonen. I tillegg er det mindre arbeid å sette seg inn i sin egen mikrotjeneste enn å forstå hele applikasjonen, noe en ofte måtte i monolittisk utvikling.

2. Hver del kan rulles ut og skaleres individuelt

   Om en f.eks. plutselig får bruk for flere ressurser på front-end så kan ressurser lett allokeres til å raskt sette opp flere tjenere. Det er slik horisontal skalering som gjør ressurser i skyen rimelige.

3. En feil i en av delene går ikke ut over resten av mikrotjenestene

   Om en tjeneste plutselig går ned vil ikke de andre delene av applikasjonen slutte å fungere. Dette gir driftere større muligheter til å reparere feil eller rulle tilbake til tidligere versjoner.

4. En kan fritt velge hvordan en utvikler sin del

   Som utvikler har man muligheten til å velge hvilket programmeringsspråk og verktøy en ønsker å bruke uten at det går ut over utviklerne på de andre mikrotjenestene.

Store aktører som Amazon, Google, LinkedIn, PayPal, Spotify, Twitter og Netflix har allerede implementert moderne oppsett med mikrotjenester, der sistnevnte har publisert overgangsprosessen sin til offentligheten. Den store kostnadsreduksjonen for IT-drift, samt lettere og raskere DevOps har gitt containerløsninger en sterk tilstedeværelse på markedet. Containerteknologi må ikke nødvendigvis skje i forbindelse med mikrotjenester, men de monolittiske løsningene kan ikke tilby mye av det som gjør containere praktisk og effektivt satt i mikrotjenester.

Orkestrering av containere

Orkestrering av containere omhandler selve håndteringen av containere, og da spesielt i store mer varierende omgivelser. I et mikrotjeneste-miljø vil en ofte ha store mengder med containernoder en skal administrere, og dette ville tatt alt for lang tid å gjøre manuelt og reaktivt. Derfor brukes dynamisk orkestrering for å løse forskjellige oppgaver, deriblant følgende:

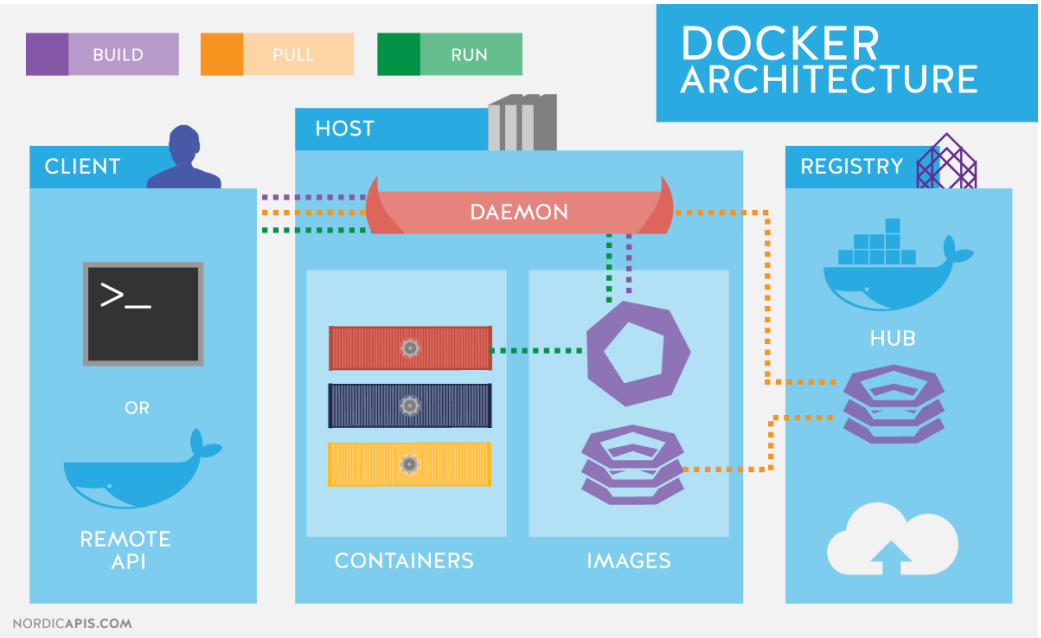
- Sette av ressurser til, og rulle ut containere.
- Tilgjengelighet av containere.
- Oppskalering og nedrulling av containere for å spre arbeidsmengden utover vertsmaskinene.

- Feiltoleranse ved at containere flyttes om en vertsmaskin går ned, eller det blir en mangel på ressurser.
- Tilgjengelighet til tjenester på containere.
- Overvåkning av containerne.

De mest populære verktøyene for orkestrering av containere er Docker Swarm og Kubernetes.

6.1.3 Docker

Docker er et verktøy som gjør det enkelt å opprette, rulle ut og kjøre applikasjoner på containere, og er i dag den mest populære plattformen innen OS-virtualisering. Programvaren ble lansert i 2013 for Linux, og kom senere ut til Windows Server i 2016. Programvaren er et open-source prosjekt utviklet av Docker Inc. med støtte fra blant annet Red Hat, Microsoft og IBM. Ved Dockers produktlansering hadde selskapet bygd det beste containerproduktet med et godt etablert økosystem, noe som resulterte i at mange bedrifter valgte å implementere teknologien deres.



Figur 2: Oppbygningen av docker arkitekturen.
Kilde: (van der Mersch, 2018)

Docker Hub - Skybasert repository hvor Docker-brukere oppretter, tester, lagrer og distribuerer container images

Docker Image - Et snapshot eller en template en bruker for å rulle ut nye containere. Et image vil være read-only, men selve containeren kan modifiseres. Endringer vil forsvinne fra containeren når den fjernes om det ikke lagres til et nytt image.

Docker Daemon - Fungerer som et mellomledd for all kommunikasjon til og fra containere. Når en bruker starter opp en container fra sin kommandolinje (docker run), så oversetter docker kommandoen til et HTTP API-kall som sendes til Docker Daemon. Daemon evaluerer videre forespørselen og kommuniserer med hostens operativsystem for å tildele containeren. Enkelt sagt er Daemon "hjernen" bak operasjonen.

Docker Client - Bruker Docker gjennom et CLI (Command Line Interface) eller med konfigurerte API-kall.

På Windows kan Docker kjøre både Linux og Windows containere på samme maskin ved hjelp av LCOW (Linux Containers on Windows). Per i dag finnes det ingen gode løsninger å kjøre Windows containere på Linux-systemer. Docker har nylig gått i et samarbeid med Microsoft for å utvikle en effektiv containerløsning for Windows; Nano Server.

6.2 Windows Server 2019 Nano Server

Høsten 2018 lanserte Microsoft tre nye versjoner av Windows-imagene sine, av disse er det Windows Nano Server 2019 prosjektgruppen skal se nærmere på. Oppdateringen Nano Server fikk fra 2016-versjonen har forsøkt å kutte alt unødvendig fra image-filen for å gjøre denne så liten og lettvektig som mulig. Dette har resultert i at den komprimerte image-filen nå er på under 100 MB.

For å få til dette er det mye som har blitt fjernet i forhold til en vanlig Windows server installasjon. OS-et er "headless", noe som betyr at det er uten mulighet for GUI og lokal innlogging på maskinen, dette betyr at all administrasjon må foregå via eksterne tilkoblinger. Det er fjernet støtte for mange funksjoner som ikke sees på som nødvendige for en container, slik som Group Policy, nettverkskort-teaming, tilgang til internett via Proxy-server, System Center Configuration Manager (SCCM) og System Center Data Protection Manager. Dette betyr at Windows Nano Server stort sett ikke kan brukes i nettverks-infrastruktur, men at en må ta i bruk Windows Server Core eller full Windows Server for disse bruksområdene. Det er mange funksjoner som

ikke er lagt til i Nano Server som standard, slik som PowerShell, .NET Core og WMI, men disse kan legges til som ekstra pakker i containeren rimelig enkelt.

Nano Server støtter ikke Windows Update, og ettersom den bruker en Host-maskins kjerne så kan den slutte å fungere hvis bare vertsmaskinen oppdateres og ikke Nano-imaget. Nano støtter ikke lenger 32-bits programvare, men det er lagt inn støtte for såkalte Risc-maskiner for bruk i Windows 10 IoT Core maskiner.

På Linux kan en finne mer lettvektige container operativsystemer, men disse kan ikke kjøre windows-tjenester. Noen eksempler på tjenester en kan kjøre på Windows Nano Containere er web-stack (WAMP/WISA) og revers prox-ytjener (Traefik).

Kort sagt, er det i Windows Nano 2019 blitt fjernet støtte for mer nettverks-infrastruktur funksjoner ettersom det da forventes at Windows Server Core blir brukt, og OS-et er optimalisert for å kjøre så fort som mulig og for å kreve minst mulig ressurser.

6.3 Bedriftsløsninger

6.3.1 Bruksmuligheter for containere

Microservice-arkitektur

Som nevnt i teori og analyse er det en hel rekke av potensielle fordeler bedrifter kan tilføye virksomheten sin ved en vilkårlig implementasjon av mikrotjenester i infrastrukturen sin. For større bedrifter kan det ligge fortjeneste i å effektivisere ressursene sine, både for deres nåværende klienter, men også for å ha både ressurser og administrativt grunnlag til å kunne ta på seg flere oppdrag og tjenesteleveringer. De sentrale fordelene med mikrotjenester for bedrifter består av:

- Skalering av tjenester – ressursbruken kan tilpasses tjenestebehovet
- Forbedret oppetid – om en tjeneste går ned så påvirker den ikke nødvendigvis de andre tjenestene i oppsettet
- Fleksibelt utviklingsmiljø - forskjellige tjenester kan programmeres i forskjellige språk grunnet mikrotjenesters API-basert samarbeid

Selv om slik innovasjon og potensiell fortjeneste er spennende er det alltid negative aspekt med implementasjon av relativt fersk teknologi:

- Ingen "best practice"

  - Teknologien er fortsatt under stor utvikling, dermed finnes det ikke en fast metode en kan følge for å sikre seg det beste mikrotjeneste-oppsettet for sin bedrift. Den store utviklingen medfører i tillegg at bedriften må være klar for å implementere forbedringer etter hvert.

- Ingen faste sikkerhetsløsninger
  - Enda en faktor å være klar over når det gjelder ny teknologi er sikkerhet. De mest etablerte sikkerhetsløsningene for containere på større skala er som regel lisensbaserte tjenester.

DevOps

Containere og mikrotjenester er som tidligere nevnt svært effektive innen DevOps-arbeid. Utrulling skjer raskere, mer fleksibelt og med mindre feil. Om en applikasjon i en container fungerer på utviklerens maskin vil den i teorien fungere likt på tjeneren.

En implementert tjeneste i containere vil ikke påvirke andre deler av oppsettet om det oppstår feil, forbedrer oppetiden på bakgrunn av raske oppdateringer og kan som nevnt skaleres etter behov. Dessuten er administrasjonen av containere praktiske og effektive, slik at lasten kan fordeles for best utnyttelse av bedriftens ressurser.

Lettvektige stacks

Et vanlig bruksområde for lettvektige stacks er webstacker basert på mikrotjenester. Dette ettersom last på webtjener kan ha stor variasjon, samt at det er relativt enkle oppgaver som utføres. I tillegg til dette krever websider høy oppetid for å unngå at lesere går andre steder. Dette oppnås ved at containere raskt kan startes opp og at de forskjellige delene av stacken kan oppdateres individuelt uten å startes om, til motsetning for monolittiske applikasjoner.

Eksempelvis har Netflix åpen dokumentasjon på hvordan mikrotjenestene deres er satt opp. Dynamikken mellom tjenestene blir oppnådd med blant annet disse verktøyene:

- Discovery av tjenster: Eureka

  Lar mikrotjenester gjøre seg selv tilgjengelige over nettverket.

- Lastbalansering: Ribbon

Leter etter services for brukere ved hjelp av info i Eureka. Om mer enn en instans av servicen er funnet lastbalanseres de av Ribbon ved å spre forespørsler til forskjellige instanser.

- Feiltoleranse: Hystix

  Bryter kontakten med ufunksjonelle tjenester og videresender forespørselen til en annen enhet.

6.3.2 Administrasjon

Om en bedrift velger å satse på microservices og containere vil administrasjonen av de være kritisk for å få mest mulig av løsningen. Hovedoppgavene til et administrasjonsverktøy for større bedrifter vil være(Wright, 2017):

- Utrulling

  Ved utrulling av tjenester må administrasjonsverktøyet kunne støtte utrulling på høy og lav skala. I produksjonssammenheng vil ikke det alltid holde med et fåtall containere om gangen.

- Oppdatering og Rollback

  Verktøyet må ha effektive løsninger for både oppdatering av containere, samt tilbakerulling til tidligere versjoner.

- Lagringsplass

  Det må være mulig å gi containerne lagringsvolum over nettverket, enten ved eget oppsett eller hos diverse skytjenester.

- Kommunikasjon

  Trygg og effektiv kommunikasjon mellom containere og hoster i tillegg til API-kall. I tillegg må det være mulighet for at containere oppdager og kan ta i bruk tjenester over nettverket.

Valg av verktøy

Grunnet oppsettets basering i lokal Windows Server 2019 vil ikke skyplatformer som Amazon og dets Amazon ECS (Elastic Container Service), samt Microsoft Azure og Azure Kubernetes Service være relevant for dette forstudiet. Plattformene er per i dag svært populære om en skal basere tjenestene sine hos skyleverandører.

Docker Swarm

Docker Swarm er en relativt enkel open-source container-administrator som er bygget rett inn i Dockers CLI (Command Line Interface) og i Docker Engine. Verktøyet står blant annet som default i Docker Enterprise. Hovedoppgaven dens er å bl.a. administrere grupper med containere som en enkeltenhet, samt orkestrering av arbeidsmengden. Problemet med Docker Swarm i bedriftssammenheng er at verktøyet ikke støtter IT-drift og overvåking på enorm skala. Den største skaleringstesten gruppen kunne finne bekreftet at Swarm klarte i 2015 å administrere 1000 noder (hostmaskiner) med 30 000 containere på 1 Swarm manager(Luzzardi, 2015). Basert på undersøkelsen egnes Docker Swarm bedre til testmiljø og virksomheter som ikke har behov for enorm skalering. Verktøyet er på ingen måte blitt gitt opp på av utviklerne, så det blir spennende å se om de vil forsøke å kjempe mot konkurrentene sine som egnes til større produksjon.

Kubernetes

Administreringsverktøyet ble i starten utviklet av Google og er nå et open-source prosjekt vedlikeholdt av CNCF (Cloud Native Computing Foundation). Per i dag er Kubernetes mest populære verktøyet for containeradministrering. I forhold til Docker Swarm har programvaren en større satsing på overvåking, vedlikehold og skalerbarhet. Kubernetes klarer å administrere hele 5000 noder med 150 000 "pods" (en eller flere containere på samme node). Programvaren støtter flere typer utvidelser som gjør verktøyet svært fleksibelt, for eksempel RedHat sin OpenShift. I forhold til de tidligere listede hovedpunktene for et administrasjonsverktøy er Kubernetes per i dag kanskje den nærmeste til å være en komplett løsning. Verktøyet har åpen kildekode på samme måte som Docker Swarm, og har et sterkt utviklersamfunn bak seg med nesten 2000 bidragsytere på GitHub. Med gode kommunikasjonsløsninger, innebygd overvåking og logging, samt mulighet for automatisk lastbalansering og skalering vil Kubernetes være et sentralt verktøy i prosjektets testmiljø.

6.3.3 Sikkerhet

Risikoer - NST SP 800-190

Det er flere nye sikkerhetsrisikoer knyttet til bruk av containere og av denne grunnen kom National Institute of Standards and Technology (NIST) i USA i September 2017 ut med en guide hvor de gikk gjennom faremomenter ved bruk av Containere. Guiden gjennomgår forskjellige områder hvor sikker-

hetshull kan oppstå, og ga i tillegg ei sjekkliste for Container-sikkerhet(Aqua, u.d.).

- Image Risks

  Ettersom Image-filene i større oppsett distribueres til hostmaskiner av et container-orkestreringssystem, er det viktig at filene ikke er satt opp med feilkonfigureringer og svakheter før de kjøres ut i produksjon. Det kan f.eks. hende imagene er feilkonfigurert med rot-tilgang, at malware har blitt lagt til på imaget, eller at det er feil i programvaren som ikke har blitt rettet opp.

- Registry Risks

  Kommunikasjon med registries (der container-image ligger) bør bare foregå over sikre kanaler, dette for å unngå man-in-the-middle angrep. Det er viktig å også passe på at bare oppdaterte images hentes fra registries til bruk, og at tilgang og rettigheter begrenses for uvedkomne.

- Orchestrator Risks

  Knyttet til administreringsverktøyene er det flere områder en må passe på. For det første må tilgang begrenses slik at hver enkelt bare får den tilgangen en trenger for å utføre arbeidsoppgaven.

  En annen risiko knyttet til dette er at tradisjonell nettverksovervåkning ikke fungerer på trafikk mellom containere/noder. Et HIPS (Host intrusion protection system) vil helle ikke fungere som med vanlige oppsett, dette ettersom verts OS-et ikke vet hva som skjer i containerne.

  I tillegg til disse grunnene kan det også være flere problemer knyttet til dårlig konfigurerte administreringsverktøy, slik som at uautoriserte tjenere blir lagt inn i et lukket cluster, eller ukryptert og uautentisert kommunikasjon som privilegert bruker.

- Container Risks

  Rapporten forteller at all kommunikasjon med utsiden kan brukes for at uvedkomne skal komme seg inn på maskinen, og at derfor disse må begrenses til bare det som er nødvendig for å drive tjenesten. Feilkonfigurering er som sagt i punktet om image-risiko også en klar trussel, med farer som tilgang til for mange systemressurser, unødvendige executable-filer på containeren, og for mye rettigheter gitt til container.

  Normalt sett vil ikke det å fikse feil direkte på containere være noen god ide, dette ettersom orkestreringsverktøyet jevnlig bytter ut containerne

med nye instanser fra imaget. Dette fører til at alle oppdateringer eller endringer på containerne utføres og legges inn på et nytt image. Dette imaget gis så til orkestreringsverktøyet for utrulling.

- Host OS Risks

  Svakheter knyttet til containere på vertsmaskinen kan skyldes at containerne har fått for mye rettigheter, det kan komme av at OS eller container-engine versjoner er utdaterte, åpne nettverksporter, dårlig konfigurert autentisering eller mangel på "Namespace Isolation" som containere er avhengig av.

  Det er også viktig å sikre vertsmaskinen mot at uvedkomne har fysisk tilgang, og at det er mange unødvendige prosesser som kjøres i bakgrunnen som kan utnyttes.

  Det er viktig å kombinere image- og host-skanning med aktiv beskyttelse og oppdaging av trusler.

Kubernetes

Tre aspekt påvirker sikkerheten i Kubernetes spesielt(Huang & Duan, u.d.):

- Ved splitting av applikasjoner til mikrotjenester sammen med enorm skalering i skyen, så øker datatrafikken som skjer internt vesentlig. Pakkede applikasjoner kan inneholde sikkerhetsfeil en ikke var klar over, interne enheter kan kompromitteres eller et phishing-angrep kan slå gjennom. Derfor er det viktig at en har strenge egress-regler for containere og maskiner internt. Administrative konsoller bør alltid passordsikres, mangelen på dette punktet gitt blant annet ut over Tesla(Duan, u.d.).
- Hver container kan ha en spesifikk komponent som er spesielt sårbar. På denne måten kan en bedrift risikere å øke antallet angrepsmetoder som kan være effektive.
- Eldre sikkerhetsmodeller kan i mange tilfeller ikke være nyttige i mikrotjeneste-oppsett. I en større operasjonell skala vil automatisering av sikkerhet ofte være en god løsning. Her er det viktig å ikke være tilfreds med det første fungerende automatiserte systemet, men å heller være frempå og videreutvikle det.

Mangel på sikring på de tre punktene kan føre til kompromitterte containere, uautorisert kommunikasjon mellom de og potensiell uthenting av sensitive data. Kubernetes har selv utgitt dokumentasjon over hvilke sikkerhetsmessige grep en burde foreta(Kubernetes, 2018). Det de velger å fremme er Transport

Layer Security (TLS) for all API-trafikk, samt autentiseringskrav for API-bruk. Videre nevnes det å begrense ressursbruk i container-clustere og å ha kontroll over hvilke rettigheter containerne har. Kubernetes og Docker kan også ta i bruk namespace-isolasjon, slik kan ikke containere påvirke prosesser i andre containere eller på hostmaskinen. I tillegg til anbefalt konfigurering har flere utviklet programvare som skal gjøre sikkerhetshverdagen enklere.

Programvare

I forbindelse med bruk av containere er det mange sikkerhetsrisikoer en må tenke på under utvikling. For å hjelpe med sikrere bruk av containere er det kommet mange verktøy som kan hjelpe med både oppsett og overvåkning. "Center for Internet Security" har for eksempel gitt ut ei liste over punkter som kan hjelpe til med å sikre mot angrep(Benjamin, 2018). For å enkelt kunne sjekke disse punktene og andre sikkerhetstrusler som nevnt i NIST-guiden(Aqua, u.d.) finnes det sikkerhetsprogramvare som Twistlock, Aqua eller andre lignende tjenester. Disse går gjennom Image-filer på vertsmaskinen for å finne svakheter og varsle om disse, den vil også sjekke innstillinger på tjenerne for å se etter kjente svakheter, og drive overvåkning av containerne for å oppdage uvanlig oppførsel.

7 Risikoanalyse

Gruppen har under gjennomgått noen forskjellige risikoer knyttet til prosjektet, og sett på sannsynlighet og konsekvens for disse. Risikoene vurderes på en skala fra 1-10 hvor ti er verst/mest sannsynlig. Videre gås det også over noen av konsekvensene hver enkelt risiko kan medføre. Avslutningsvis gås det også over mulige tiltak på en del av punktene.

- Splid i gruppe

  Sannsynlighet: 2
  Konsekvens: 9

  Kan føre til at deltakerne ikke klarer å holde oss innenfor tidsrammene på oppgaven, eller i verste fall ikke kan levere en ferdig oppgave ved slutten av prosjektet.

  Faste arbeidstider og god kommunikasjon vil være til stor hjelp ved å holde gruppen på samme side.

- For stort omfang av oppgave

  Sannsynlighet: 5
  Konsekvens: 5

  Om oppgaven blir for stor, kan dette føre til at gruppen ikke klarer å holde oss innenfor de tidsrammene som har blitt satt for prosjektet. Hvis dette utsettes for mye, kan det i verste fall føre til at ikke oppgaven kan leveres eller må leveres uferdig.

  Tiltak mot konsekvensene her vil være å kartlegge hva som er viktigst og mest sentralt med oppgaven, og bygge videre på den derfra.

- Sykdom

  Sannsynlighet: 5
  Konsekvens: 3

  Sykdom kan føre til utsettelser i prosjektet, noe som enten fører til utsettelser i forhold til tiltenkte planer, mer arbeid utenom planlagte tider, eller i verste fall at ikke gruppen klarer å holde seg innenfor tidsrammene gruppen selv har satt på prosjektet.

  Det er få hensiktsmessige tiltak en kan gjøre her som ikke er knyttet til livsstil. På lengre tidsperspektiv kan en prøve å holde seg til et sunt kosthold og jevnlig trene.

- Brudd på timeplan

  Sannsynlighet: 5
  Konsekvens: 3

  Hvis deler av prosjektet tar mer tid enn ventet, må dette enten tas igjen senere i prosjektet, eller det må jobbes mer utenom planlagte timer for prosjektet.

  Ved å planlegge nøye, og sette mindre tidsfrister oftere kan en tidlig vite om en er bak skjema, og på den måten få gjort noe med etterslepet tidlig. Dette vil da forhindre de verste konsekvensene knyttet til punktet.

- Mangelfull dokumentasjon av programvare

  Sannsynlighet: 6
  Konsekvens: 4

  Ved mangelfull dokumentasjon av programvaren brukt i oppgaven kan det komme utsettelser, og mer tid brukt på oppsett av testlab enn hva som er planlagt. Grunnet Nano Server fortsatt er relativt fersk i containerverdenen når gruppen skriver oppgaven så vil dokumentasjon og støttende programvare ikke alltid være tilgjengelig.

  Lite en kan gjøre for å forhindre at dette skjer, men ved å velge programvare som er viden brukt er sannsynligheten mindre for at dokumentasjonen er mangelfull.

- Maskinvareproblemer

  Sannsynlighet: 2
  Konsekvens: 6

  Konsekvensene av maskinvareproblemer vil kunne variere stort med tank på hva som feiler, og når det går galt. En tidlig maskinvarefeil kan forsinke prosjektet noe pga. feilsøking, mens en feil sent i prosjektet kan være et stort problem ettersom data og mye tid kan gå tap. Det beregnes som relativt lite sannsynlig at noe går galt med selve maskinvaren underveis i oppgaven.

  For å forhindre tap av data ved maskinvarefeil er sikkerhetskopiering av data viktig.

8 Retningslinjer og standarder

8.1 Krav til dokumentasjon

Følgende dokumentasjon skal leveres i løpet av bacheloroppgaven:

Tabell 1: Dokumentasjonskrav

Navn	Innleveringsdato
Forstudierapport	25.01.19
Designdokument	08.02.19
Driftsdokument	19.04.19
Sluttrapport	26.04.19
Egenvurdering	01.05.19
Endelig levering	20.05.19

Endelig innlevering skjer elektronisk i Inspera, der vedlegg og rapporter samles i en PDF-fil. Dokumentasjon og rapporter underveis godkjennes av veileder og er tilgjengelig for veilederne på gruppens Sharepoint-side.

8.2 Krav til kvalitetsgjennomgang

Kvalitetssjekk gjennomføres av veileder og samtlige prosjektdeltakere etter levering i henhold til tabell 1. Tilbakemeldingene gruppen mottar vil være grunnlaget for revisjon av dokumentene. Rapportene vil sammen med prosjektets gjennomførsel være vurderingsgrunnlaget for bacheloroppgaven. I tillegg vil presentasjon av oppgaven og forholdet til tidsfrister påvirke vurderingen i mindre grad.

9 Prosjektorganisering

Prosjektgruppen består av to prosjektdeltakere som utfører bacheloroppgaven. Gruppen har tre veiledere til oppgaven, der en av de står for kvalitetskontroll. I tillegg stiller Basefarm med en teknisk konsulent under prosjektgjennomførselen.

Figur 3: Prosjektorganisering

10 Anbefaling om videre arbied

Containerteknologi har de siste årene blitt tatt i bruk av mang av de største bedriftene i verden, og har vist seg å gi flere fordeler, blant annet lav ressursbruk og muligheten til å lett kunne oppskaleres.

Teknologien virker lovende og kan bidra til økt fortjeneste. Med de fordelene gruppen har sett knyttet til teknologien anbefales det å videreføre arbeidet med de rammene og planene som er lagt frem i denne rapporten. Mot slutten av prosjektet vil prosjektgruppen komme med videre anbefalinger om hvordan det fremstilte oppsettet burde videreutvikles for å kunne nyttes i IT-tjenestelevering.

Referanser

Aqua. (u.d.). *Practical guide: Nist special publication 800-190 application container security guide.* Hentet 15.01.2019 fra https://cdn2.hubspot.net/hubfs/1665891/Aqua%20NIST%20Guide.pdf

Benjamin, P. (2018, 07). *Docker security best-practices.* Hentet 21.01.2019 fra https://dev.to/petermbenjamin/docker-security-best-practices-45ih

Chamberlain, D. (2018, 03). *Containers vs. virtual machines.* Hentet 27.04.2019 fra https://blog.netapp.com/blogs/containers-vs-vms/

Duan, G. (u.d.). *Cryptojacking and crypto mining – tesla, kubernetes, and jenkins exploits.* Hentet 17.01.2019 fra https://kubernetes.io/docs/tasks/administer-cluster/securing-a-cluster/

Huang, F. & Duan, G. (u.d.). *The ultimate guide to kubernetes security.* Hentet 24.01.2019 fra https://neuvector.com/container-security/kubernetes-security-guide/

Janssen, T. (2018, 02). *What is cloud-native? is it hype or the future of software development?* Hentet 16.01.2019 fra https://stackify.com/cloud-native/%7D

Kelly, D. (2017, 11). *Monolithic vs microservices.* Hentet 16.01.2019 fra https://blog.containership.io/monolithisvsmicro/

Kubernetes. (2018, 01). *Securing a cluster.* Hentet 18.01.2019 fra https://kubernetes.io/docs/tasks/administer-cluster/securing-a-cluster/

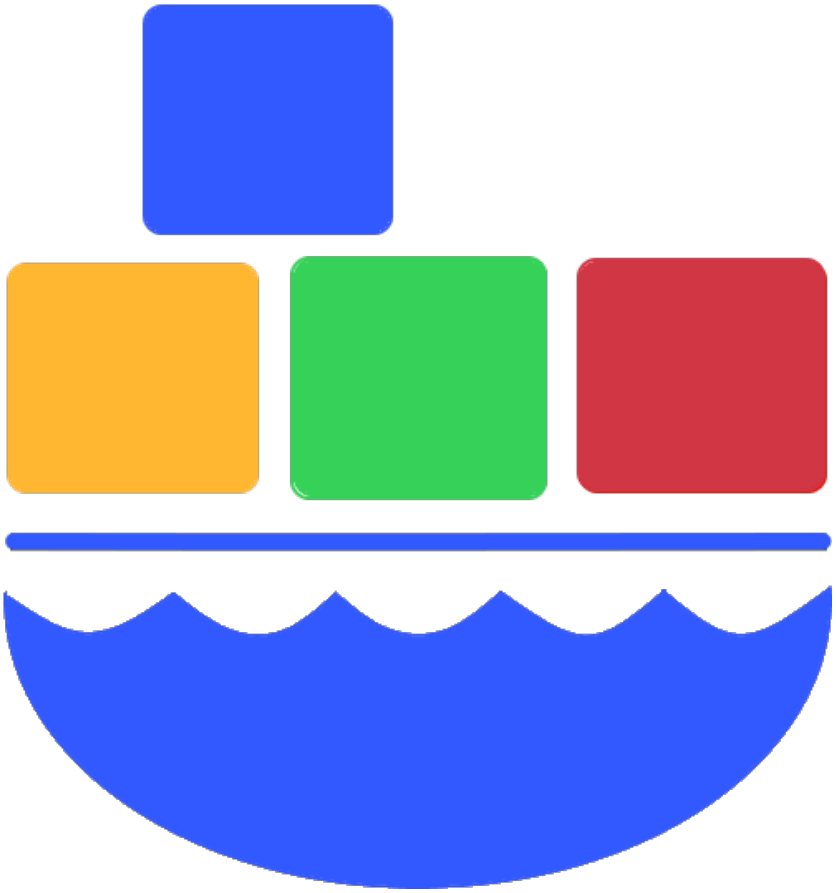
Luzzardi, A. (2015, 11). *Scale testing docker swarm to 30,000 containers.* Hentet 21.01.2019 fra https://blog.docker.com/2015/11/scale-testing-docker-swarm-30000-containers/

Osnat, R. (2018, 03). *A brief history of containers: From the 1970s to 2017.* Hentet 15.02.2019 fra https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016

van der Mersch, V. (2018, 03). *Docker architecture.* Hentet 27.04.2018 fra https://nordicapis.com/api-driven-devops-spotlight-on-docker/

Wright, C. (2017, 06). *Kubernetes vs docker swarm.* Hentet 21.01.2019 fra https://platform9.com/blog/kubernetes-docker-swarm-compared/

Containeroppsett i Windows Server 2019



Designrapport

Anders Kvanvig
andekva@stud.ntnu.no

Adis Pinjic
adisp@stud.ntnu.no

Revisjonshistorie

Versjon	Dato	Forfatter(e)	Beskrivelse
1.0	14.02.19	Adis Pinjic, Anders Kvanvig	Første utkast fra prosjektgruppen.
1.1	15.04.19	Adis Pinjic, Anders Kvanvig	Andre utkast, gjennomgått dokument og fikset opp i diverse mangler
2.0	27.04.19	Adis Pinjic, Anders Kvanvig	Lagt rapport inn i LaTeX

Innhold

Figurer

1 Innledning

Rapporten vil gi en tekstlig beskrivelse av planlagt oppsett brukt i oppgaven. Det vil bli gjennomgått hvilke verktøy og løsninger som skal tas i bruk, begrunnelsen for hvorfor de ble valgt og hva de skal oppnå på et teknisk nivå. Dokumentet er oppdelt i:

Dokumentets hensikt – En kort beskrivelse av målene til prosjektet og hva designapporten vil tilby av informasjon.

Avgrensing – Hvilke avgrensinger prosjektdeltakerne selv har satt samt ytre avgrensinger som minsker omfanget av oppgaven.

Definisjoner og forkortelser – Ordliste og forklaring av begrepene som blir brukt i rapporten.

Kort om oppdragsgiver og behov – Hvilke forutsetninger oppgaven har i forhold til oppdragsgiver.

Tekniske løsninger – Både en overordnet og en detaljert beskrivelse av elementene i prosjektets oppsett.

2 Dokumentets hensikt

Hensikten med prosjektet er å tilby bedrifter den tekniske innsikten i containeroppsett i Windows for å kunne bedømme dets nytte for deres egen virksomhet. Denne rapporten vil bidra til å gi innsikt i:

- Hvilke verktøy som ble valgt for prosjektets testmiljø
- Bakgrunnen for hvorfor verktøyene ble valgt
- Hva verktøyene vil oppnå i praksis

3 Avgrensninger

Det har i oppgaven blitt lagt flere avgrensinger for å unngå at omfanget blir for stort. Den første av disse er å ikke ta i bruk skytjenester. Skyplattformer som Azure, AWS og Google Cloud har godt integrerte hjelpemiddel for containere, men det ville ha gjort oppgaven mer komplisert og tidkrevende. Tiden som kreves for å gi bachelorgruppen den nødvendige opplæringen for å ta tjenesten i bruk ville kanskje gått ut over kvaliteten på oppgaven.

En videre innskrenkning prosjektgruppen har gjort er å holde oppgaven på Windows-plattformen. Containere har i flere år blitt brukt på Linux, og det kunne derfor vært interessant å sett hvordan Windows Nano Server gjør det i forhold til liknende Linux oppsett. Dette ville igjen vært tidkrevende, og er dermed noe prosjektgruppen valgte å droppe. I tillegg er det noen av verktøyene som ikke er mulig å replikere på Windows per i dag, en av disse er maskinrollen som Kubernetes Master.

Både sikkerhetsbildet og driftsoppsettet blir påvirket av hvor ny bruken av Windows-containere er på markedet. Containerløsninger har per i dag mangel på dokumentasjon som standarder og retningslinjer. Dette resulterer i at gruppens oppsett har ingen garanti for å være det mest optimaliserte i forhold til stordrift av mikrotjenester i Windows.

Gruppens oppsett vil foregå på en relativt liten skala, og testene vil gjennomføres på samme maskinvare. På grunn av dette vil ikke dette prosjektet egnes til å bedømme containeres effektivitet i et reelt produksjonsmiljø, men heller som en demonstrasjon av teknologiens egenskaper og potensial.

Det siste punktet for avgrensing er Nano Server sin umodne status på Windows 1809. Flere tjenester som var mulig å kjøre på Nano-servere i tidligere versjoner av Windows har ikke enda blitt ført over til 1809 ved gjennomføringen av prosjektet. Dette resulterer i at prosjektgruppen må oftere velge et større container-image (Server Core) for tjenestene som blir kjørt.

4 Definisjoner og forkortelser

- Kubernetes:
  - K8s:

    Kortform for Kubernetes. Kommer av at bokstavene i midten byttes med antall tegn mellom ”k” og ”s”.

  - Pod:

    En pod er et sett med kjørende containere (en eller flere) som skal holdes samlet. Pods blir kjørt på nodene i et cluster.

  - Service:

    Definerer et sett med pods og hvordan de skal nås over nettverket. Dette opprettes på grunn av pods skaleres regelmessig, og da trenger man en fast oversikt over hvor man kan nå tjenesten.

  - Node:

    En fysisk eller virtuell maskin i et Kubernetes cluster.

  - Cluster:

    Flere noder som styres av Kubernetes.

  - Kubelet:

    Agent som kjører på hver enkelt node i Kubernetes clusteret, denne utfører oppgaver på vegne av Kubernetes-masteren på nodene. Eksempler på oppgaver kan være å sette opp en ny pod eller liknende.

  - Kubectl:

    Kommandolinjen til Kubernetes som installeres på Kubernetes Master.

  - Kube-Proxy:

    En ressurs som kjøres på hver node som inneholder regler for IP-tables. Tjenesten gjør i tillegg tilfeldig lastbalansering mellom de benyttede applikasjonene.

  - Master:

    Tjener som administrerer Kubernetes-cluster. Denne vil inneholde en scheduler for å velge hvor pods skal kjøres, etcd for å lagre data

om clusteret, en API-server for å kunne kontrollere clusteret, samt kontrollører for en del andre funksjoner i Kubernetes.

- Microservice:

  Er en måte å utvikle programvare på hvor programmet bygges opp av flere mindre og løst sammenkoblede tjenester. Dette blir i stedet for den mer tradisjonelle oppbyggningen hvor det er en mer monolittisk oppbygning.

- Webstack:

  Referer til alle komponentene som trengs for å sette opp ei webside. Eksempler på dette kan være LAMP (Linux, Apache, MySQL og PHP) på Linux eller WIMP (Windows, IIS, MSSQL og PHP) på Windows.

5 Kort om oppdragsgiver og behov

Prosjektgruppens oppdragsgiver er Basefarm AS. Bedriften ønsker å få innsikt i bruksmulighetene til containere i Windows Server 2019 gjennom gruppens bacheloroppgave. Gjennom dialoger med gruppens veiledere har deltakerne og interessentene kommet frem til et hensiktsmessig oppsett som skal opprettes på en fysisk server.

5.1 Hvorfor valg av produkt-teknologi-løsning

Det eneste programvarekravet prosjektgruppen fikk tildelt var at containerløsningen skal settes opp i Windows Server 2019 med hjelp av Windows Nano Server. Det resterende rammeverket var opp til prosjektgruppen selv. Generelt ble programvare bestemt basert på hvor godt etablert programmet var, hvor mange brukere applikasjonen har, samt om alle behovene for oppsettet ble dekket. Prosjektoppgaven hadde ikke et forbruksbudsjett, dermed er de fleste applikasjonene basert på åpen kildekode og prøveversjoner. Aspektet med oppgaven som blir mest rammet på bakgrunn av dette er sikkerhet. Dette grunnet de mest etablerte programvarene for sikkerhet og monitorering er lisensbaserte.

5.2 Prosjektets oppsett

Docker

Docker ble valgt som en ressurs for oppgaven på bakgrunn av plattformens anerkjennelse og popularitet innen containermiljøet i IT-næringslivet. Denne statusen på markedet medfører at det vil være enormt med ressurser tilgjengelige for gruppen for å integrere verktøyet i prosjektets testlab. Dessuten har Docker åpen kildekode som gjør at prosjektgruppen kan bruke verktøyet gratis.

Kubernetes

Etter observasjon av hvilke administreringsverktøy som brukes i kombinasjon med containere ble det tidlig tydelig at Kubernetes var klart det mest brukte i bedriftssammenheng. Programvaren støtter Windows Server 2019, har åpen kildekode og god nok dokumentasjon til at det kan implementeres i testmiljøet til oppgaven. Ut ifra hva prosjektgruppen har funnet i forstudierapporten skal Kubernetes være ideell for skalerbarhet, lastbalansering

og overvåkning. Verktøyet vil i teorien kunne oppfylle de fleste administrasjonsønsker en virksomhet vil ha til sitt driftsmiljø.

Webstack

Prosjektgruppen ønsker å demonstrere containerteknologien i et miljø der de er mest effektiv og relativt godt etablerte. Å tilby webtjenester er en krevende oppgave som blir mer effektiv ved å baseres på containere og mikrotjenester. Ved å bruke et slikt oppsett for testlabben vil prosjektgruppen kunne sette opp realistiske scenario for lastbalansering, automatisk skalering og overvåking. I testlabben vil en videreutvikling av Microsoft sin containerwebserver basert i PowerShell1 være utgangspunktet for en webstack-demo.

TICK-stack

En TICK-stack er en open-source overvåkningsplattform som bl.a. benyttes av Basefarm. Stacken består av fire forskjellige komponenter:

- Telegraf

  Innhenting av data fra vertsmaskinene.

- InfluxDB

  Mottar, holder styr på og gjør data fra Telegraf tilgjengelig for andre tjenester.

- Chronograf

  Tilbyr et grensesnitt for bruk av data i InfluxDB.

- Kapacitor

  Står for prosessering av data og varsling ved eventuelle problemer.

Ved hjelp av TICK-stacken ønsker gruppen å sette opp overvåkning av vertsmaskinene og containerne, noe som nærmere forklares under detaljert løsningsbeskrivelse.

6 Tekniske Løsninger

6.1 Overordnet beskrivelse

Prosjektets testmiljø vil bruke Docker Community Edition som containerplattform på Windows Server 2019 og Ubuntu Server 16.04. Tjenester vil kjøres i Windows-containere (Nano Server og Server Core) og administreres ved hjelp av en Linux Kubernetes Master som styrer Kubelet på hver node. Master-noden vil i tillegg kjøre tjenester som i dag bare kan kjøres på Linux. Overvåking av verktøy og applikasjoner gjennomføres med TICK-Stack. Databasen, GUI og varselssystemet til TICK kjøres på master-noden, i tillegg til at Telegraf er en agent på Windows-nodene som henter ressursbruk.

6.2 Funksjonelle behov

Docker

- OS-virtualisering
- Opprette Docker-Images for applikasjoner som inkluderer programvaren og støttebibliotek

Kubernetes

- Rulle ut tjenester som pods på noder i clusteret
- Skalering av tjenester
- Lastbalansering av tjenester
- Oppdatering av tjenester

Webapplikasjon

- Delt opp i mikrotjenester
- Kompatibelt med containeroppsett

Sikkerhet

- Riktig oppsett av brannmur, nettverkskonfigurasjon og retningslinjer
- Hensiktsmessige rettigheter for noder og containere
- Tjenere rulles ut med bare den programvaren som er nødvendig

Overvåking

- Varsle IT-administrasjon om oppsettet trenger menneskelig interaksjon

- Statistikk over oppetid
- Kan implementeres med Windows-noder
- Overføre hensiktsmessig informasjon til Kubernetes

6.3 Konfigurering og krav til driftsrutiner

- Docker må installeres på hver node for at containere skal kunne kjøres
- Containere må rulles ut med nødvendig programvare og støttebibliotek for at applikasjonen skal kunne kjøres
- Krav for skalering og lastbalansering må konfigureres i Kubernetes
- Overvåking og varsling konfigureres gjennom Chronograf-vinduet

6.4 Sikkerhetsvurdering

Prosjektets testmiljø vil i utgangspunktet ikke være tilgjengelig for offentligheten og vil være på et internt nettverk. På bakgrunn av dette vil oppsettet mest sannsynlig ikke måtte motstå eksterne angrep over nettet. Likevel vil prosjektets fokus på containerdrift for større bedrifter medføre at oppsettet blir utviklet til å kunne være rustet mot sikkerhetstrusler. Mikrotjenester resulterer i større menger horisontal nettverkstrafikk mellom tjenester i forhold til vanlige monolittiske oppsett. Dette resulterer i at et større fokus vil settes på at trafikken mellom tjenerne er sikret og at containerne er konfigurert riktig.

Containeroppsett og mikrotjenester er som nevnt fortsatt i et relativt umodent stadium. Sikkerhetsaspektet i slike miljø kan fort bli usikre og potensielt utnyttet, spesielt på grunn av mangelfull sikkerhetsdokumentasjon som f.eks. ISO-standarder. Prosjektgruppen vil basere sikringen av oppsettet på NIST sin sikkerhetsveiledning fra 2017 kalt "NIST Special Publication 800-190 Application Container Security Guide"(Souppaya, Morello & Scarfone, 2017). I tillegg vil Kubernetes sin egen sikkerhetsdokumentasjon nyttes for å få innsikt i hvordan en sikrer et cluster fra inntrengere.

7 Detaljerte løsningsbeskrivelser

7.1 Om testmiljøet

I testmiljøet planlagt for denne oppgaven vil prosjektgruppen ha alt kjørende på en fysisk maskin. Alle maskinene nevnt nedenfor skal kjøre som VM-er på denne fysiske maskinen. Oppsettet skal inneholde to forskjellige typer maskiner:

- Windows Server 2019-maskiner
  - Er en arbeidsnode i Kubernetes-clusteret
  - Docker
  - Kubelet
  - Kube-Proxy
  - Flannel
  - Containerapplikasjoner
  - Telegraf (TICK)
- Ubuntu 16.04-tjener
  - Er en Kubernetes Master-node
  - Flannel
  - Kube-DNS
  - Kube-Proxy
  - MetallLB
  - Metrics-server
  - Telegraf (TICK)
  - InfluxDB (TICK)
  - Chronograf (TICK)
  - Kapacitor (TICK)

Server 2019-maskinene vil være selve nodene på Kubernetes sitt cluster som vil kjøre applikasjonene Kubernetes ruller ut fra master-maskinen. Den vil være utstyrt med Docker for å kunne virtualisere OS-et sitt og kjøre containere, samt Kubelet for å kunne ta imot kommandoer fra Kubernetes. Maskinen

vil på samme måte som alle noder i et Kubernetes-cluster være utstyrt med Kube-Proxy for IP-tables for Kubernetes sine applikasjoner. I tillegg til å kjøre prosjektets webserver-applikasjon vil noden ha Telgraf installert for å sende ressursbruk til InfluxDB og resten av TICK-stacken.

Ubuntu-maskinen vil være Kubernetes-clusterets master. Det er denne maskinen som bestemmer alt rundt utrulling av applikasjoner i Kubernets, f.eks. hvor den skal kjøres, hvor mange instanser og hvordan den skal nås. Applikasjoner vil automatisk kunne få eksterne IP-adresser med den lokale lastbalansereren MetalLB så lenge servicen er av typen "LoadBalancer". Helm installeres og kan brukes til å rulle ut rulle ut ferdige applikasjoner. Og til slutt kjører masteren tre av komponentene i TICK-stacken, noe som vil forklares nærmere i punkt 7.5.

7.2 Docker

Docker settes i testmiljøet opp på Windows-tjenerne og master-maskinene, og brukes til å virtualisere OS-et til maskinen for å kjøre containere. Programvaren gir containerne tilgang til systemets ressurser, og har også ansvaret for å begrense denne tilgangen etter hvilke regler som er satt for disse. Systemkall sendt av containerne vil sjekkes mot de rettighetene som er oppgitt og enten godkjennes og gjennomføres av Docker, eller avslås.

Docker blir i praksis for prosjektet et verktøy for oppretting og oppbevaring av image-filer for containere. Utvikling i Docker vil som regel ikke foregå på maskinnodene som kjører applikasjonene. Det anbefales å bruke lokale installasjoner av Docker på egne maskiner til å utvikle containerene som rulles ut i testlabben.

7.3 Kubernetes

En god analogi for Docker og Kubernetes er en vanlig privatbil. Docker er bilmotoren som står bak container-images, mens Kubernetes er i førersetet i bilen. I vanlig drift vil man som i en bil vanligvis forholde seg til rattet, styringen og annen administrasjon. Av og til må en naturligvis åpne panseret til Docker og lage nye eller endre eksisterende images.

I praksis betyr dette at Kubernetes vil være verktøyet som tar seg av:

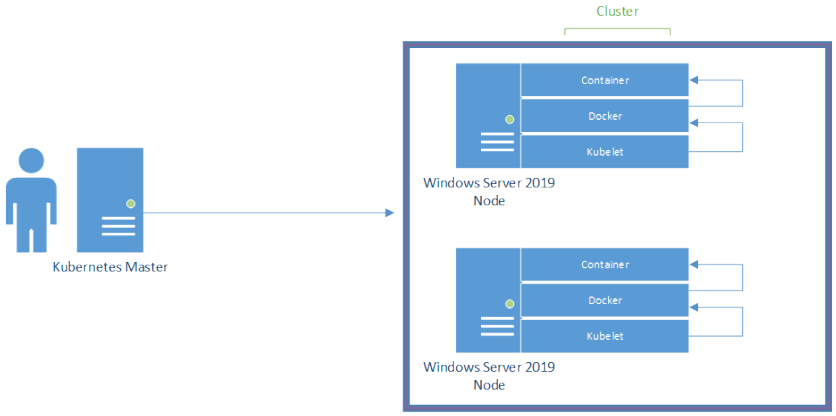
- Nettverk og kommunikasjon
- Utrulling og tilbakekalling av applikasjoner
- Lastbalansering
- Skalering
- Oppdatering

Selv om det er store bruksmuligheter for Kubernetes bør man ikke tenke på det som en tradisjonell PaaS (Platform as a service). Kubernetes gir byggeklossene for å lage en administrasjons- og utviklingsplattform der alle default-løsninger er valgfrie. Noen bruksområder som kan misforstås:

- Bygger ikke applikasjoner – det gjøres i Docker
- Inkluderer ikke middleware som webservere og databaser, men slikt kan kjøres i Kubernetes
- Dikterer ikke logging og overvåking, men kan kjøres i Kubernetes (i testmiljøet løses dette med TICK-stack)
- Tar seg ikke av maskinkonfigurasjon, vedlikehold og feilretting av host-maskinen

Kubernetes i testmiljøet kan deles i to; administrering og praksis.

Administrasjon – hvordan Kubernetes håndteres:



Figur 1: Enkel fremstilling av planlagt cluster

Når Kubernetes-oppsettet er ferdig skal gruppens administrasjon skje fra en Kubernetes Master-maskin. I prosjektets tilfelle vil masteren være på et Linux-OS siden mastermaskiner ikke er støttet på Windows enda. Masteren vil i praksis være arbeidsstasjonen for en IT-admin der all konfigurering, monitorering og administrasjon foregår. Det masteren skal håndtere er forskjellige clustere. Et Kubernetes-cluster inneholder noder som kjører selve applikasjonene. Oppsettet på nodene blir først en Kubernetes-agent kalt Kubelet som behandler noden på vegne av Kubernetes sin konfigurasjon. Deretter kommer Docker som står for OS-virtualiseringen på noden, og til slutt selve applikasjonen i en container.

Kubernetes vil også stå for brannmur-tjenester i prosjektets oppsett for å hindre uønsket nettverkskommunikasjon. Som standard vil containerne bli satt til å godta alt, men dette skal her endres til å bare godta trafikk fra visse avsendere, på visse porter for å minke angrepsflaten. For at dette skal kunne implementeres må K8s settes opp med et network-plugin. Planen blir her å gå for utvidelsen Flannel ettersom denne støtter for bruk av Windows-tjenere(Kubernetes, 2019).

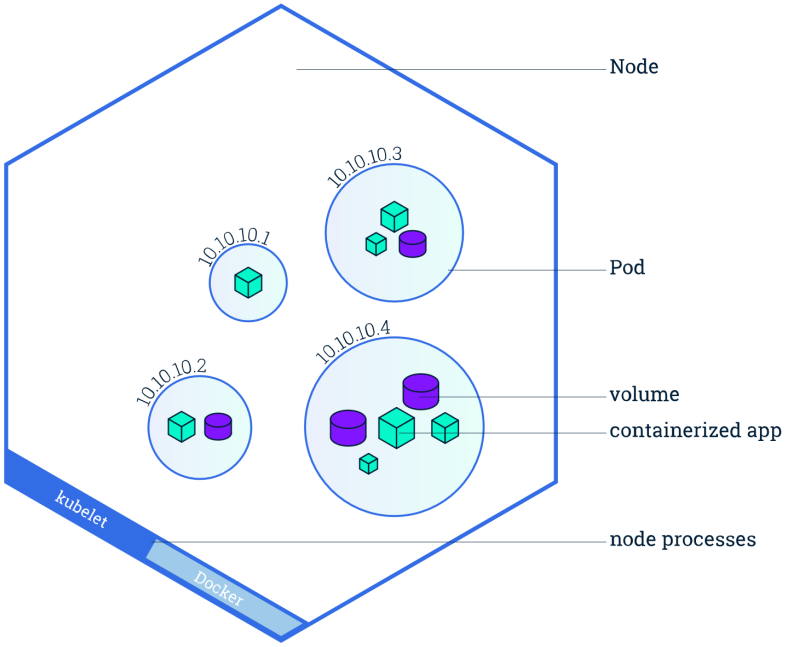
Lastbalansering vil skje på to nivå: lastfordeling mellom pods og lastfordeling av hvordan en når tjenesten. Kube-Proxy er installert på hver node og fordeler lasten jevnt mellom instansene av en tjeneste. I praksis betyr dette at om en har for eksempel ti webtjenere, så vil en heller fordele litt last på hver av dem i stedet for full last på tre instanser. Når det gjelder lastbalansering av eksterne tilkoblinger så kommer MetalLB inn på bildet. Verktøyet er utviklet for lokal drift av Kubernetes og vil fordele eksterne IP-adresser til tjenester

automatisk.

Autoskalering av tjenester vil implementeres med Kubernetes sin egne "Horizontal Pod Autoscaler" (HPA). HPA krever en "metrics-server" på clusteret og vil skalere tjenestene basert på hvilke forespørsler og grenser som er satt for tjenesten.

Utrulling av tjenester gjennomføres gjennom kommandolinjen til Kubernetes kalt "kubectl". Herfra kan en enten rulle ut applikasjonen med hjelp av konfigurerte YAML-filer som beskriver all info en trenger for å kjøre en tjeneste i et Kubernetes-cluster. Dette inkluderer hvor Docker-imaget hentes fra, hvilken type tjeneste det er, hvilken type node skal kjøre den og lignende. Alternativt kan verktøyet Helm brukes for å rulle ut tjenester, men dette verktøyet benytter også YAML-filer. Det positive med Helm er at den tilbyr større kompleksitet for utrullingene, der man blant annet kan gjennomføre versjonskontroll og tilbakerulling.

Praksis – hvordan pods faktisk fungerer:



Figur 2: Forenklet fremstilling av en node i Kubernetes
Kilde: (Kubernetes, 2018)

Tjenester blir delt inn i pods som kjøres på nodene, slik opprettholdes et mikrotjenestemiljø. En pod er en gruppe med containere som rulles ut på samme node. I prosjektets tilfelle kunne gruppen f.eks. lagt ved en database i samme pod som webapplikasjonen. Hver pod vil som presentert i figuren over ha sitt eget lagringsvolum og en applikasjon pakket i en container. Når Kubernetes blir nødt til å skalere tjenesten vil verktøyet rulle ut nye pods i samme clusteret. Det er her effektiviteten kommer spesielt inn, for om det er en spesifikk mikrotjeneste som får stor etterspørsel så kan Kubernetes skalere den enkelte tjenesten.

7.4 Webapplikasjon

Prosjektgruppen har utviklet en webapplikasjon som bygger videre på Microsoft sin "win-webserver"(Microsoft, 2018), som er en enkel webtjeneste utviklet i PowerShell. Webapplikasjonen vil bestå av tre mikrotjenester:

- Et grensesnitt for brukeren
- To tjenester tilgjengelig med API-kall

Grensesnittet vil gi brukeren et enkelt valg mellom de to tjenestene applikasjonen tilbyr. Tjenestene vil svare på enkle webforespørsler som blir behandlet av grensesnittet. Dette betyr at tjenestene ikke vil være direkte tilgjengelige for brukeren og vil bare kunne nås internt. Det at applikasjonen er delt i mikrotjenester gjør at den vil kunne tilrettelegge testing innen skalering og lastbalansering.

7.5 TICK-stack

I oppsettet vil loggføring og overvåking foregå i en såkalt TICK-Stack. Tjenesten er oppbygd av fire open-source prosjekt kalt Telegraf, InfluxDB, Chronograf og Kapacitor.

Telegraf plasseres på hver node som skal overvåkes og samler informasjon fra Docker, Kubernetes og operativsystemet til noden. Dataen sendes så videre til InfluxDB som lagrer infoen i sin database, som videre kan nyttes av de andre komponentene. Chronograf er det administrative brukergrensesnittet som gir mulighet for monitorering med egendefinerte dashboards, samt generell konfigurering av stacken.

Avslutningsvis vil Kapacitor prosessere dataene fra InfluxDB. Verktøyet støtter både behandling av kontinuerlig strømming av data og større batch-filer. Informasjonen brukes til å varsle enheter basert på konfigurasjonen og utføring av bestemte handlinger for problemet. Kapacitor kan i tillegg integreres med kommunikasjonsplattformer som f.eks. Slack eller vanlig epost.

I prosjektgruppens testmiljø vil Telegraf inkluderes på hver node i tillegg til Kubernes master. Programmet vil så samle ressursbruk for både noden (systeminfo) og pods kjørende inn i noden. De resterende delene av stacken (Chronograf, InfluxDB og Kapacitor) vil kjøres på Kubernetes Master.

7.6 Sikkerhet

- Brannmurer mellom webtjenester og internt nett

  Kubernetes vil i kombinasjon med Flannel (En utvidelse for K8s) håndtere hvem som kan kommunisere med hvilke containere. Dette vil bli gjort ved å sette opp nettverksregler for tjenestene gjennom Kubernetes. Dette løses med "network policy" som en definerer i Kubernetes og settes i effekt av Flannel.

- Overvåking og oppdateringer av containere

  I oppsettet planlagt for denne oppgaven er målet å bruke TICK-stacken for loggføring og ressursovervåkning av noder og containere.

  For oppdatering av imagefilene oppretter en nye versjoner ved å laste inn nye versjoner av programvare på ei ny image-fil, gjerne ved hjelp av script. Disse kan så legges til i K8s og rulles ut som en ny versjon så ordner programvaren det å bytte ut det gamle imaget med den nye versjonen. I Kubernetes kalles dette "Rolling-update", noe som vil demonstreres i driftsrapporten.

- Riktig konfigurering av containere

  Containere skal ikke rulles ut i produksjon med programvare den ikke trenger. Instansen skal kun inneholde programvaren den skal kjøre og de verktøyene som er nødvendig. Dette slik at det er færre mulige angrepsvektorer for uvedkomne. Det vil også være et mål her å begrense angrepsflaten ved å riktig sette opp nettverkstilganger. Dette vil demonstreres med "security policies" en kan definere for pods.

Referanser

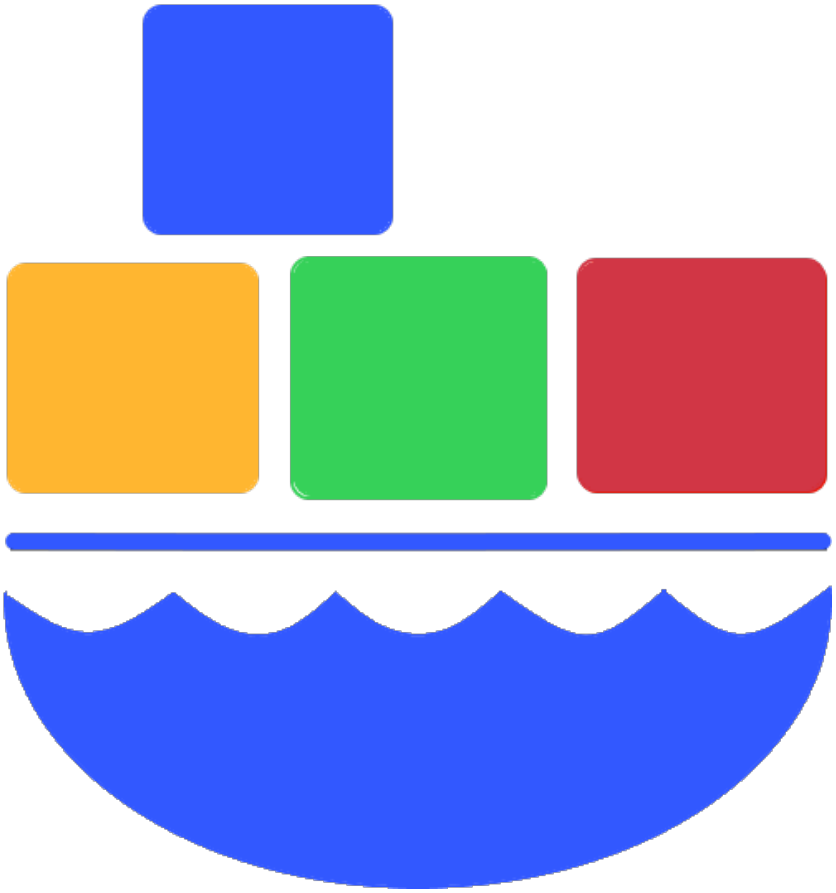
Kubernetes. (2018, 11). *Node overview.* Hentet 15.04.2019 fra https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/#node-overview

Kubernetes. (2019, 03). *Using windows server containers in kubernetes.* Hentet 27.04.2019 fra https://kubernetes.io/docs/setup/windows/

Microsoft. (2018, 03). *Webserver.* Hentet 15.04.2019 fra https://github.com/Microsoft/SDN/blob/master/Kubernetes/WebServer.yaml

Souppaya, M., Morello, J. & Scarfone, K. (2017, 09). *Nist special publication 800-190 application container security guide.* Hentet 15.04.2019 fra https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf

Containeroppsett i Windows Server 2019



Driftsrapport

Anders Kvanvig
andekva@stud.ntnu.no

Adis Pinjic
adisp@stud.ntnu.no

Revisjonshistorie

Versjon	Dato	Forfatter(e)	Beskrivelse
1.0	25.04.19	Adis Pinjic, Anders Kvanvig	Første utkast fra prosjektgruppen.
2.0	27.04.19	Adis Pinjic, Anders Kvanvig	Lagt rapport inn i LaTeX

Innhold

Figurer

1 Introduksjon

Driftsrapporten vil presentere det praktiske oppsettet i bacheloroppgaven som ble fremstilt i designdokumentet. Rapporten er todelt mellom installasjon og demonstrasjon:

Installasjon og oppsett – Beskriver hvordan hver komponent i oppsettet ble installert og konfigurert.

- **Kubernetes Master** - Guide for å sette opp en Kubernetes Master på Ubuntu 16.04
- **Flannel** - Oppsett av Flannel som nettverksløsning for Kubernetes
- **Helm** - Utrullingsprogramvare for Kubernetes-clusteret
- **MetalLB** - Lastbalanseringsverktøy for lokal Kubernetes
- **Windows-noder** - Oppsett for nodene som skal kjøre pods i Kubernetes-clusteret
- **Utrulling av tjeneste** - Hvordan rulle ut prosjektets mikrotjenester på et Kubernetes-cluster
- **Autoskalering av tjenester** - Hvordan automatisk skalering kan konfigureres med HPA
- **TICK-stack** - Hvordan overvåkningsverktøyet TICK rulles ut på et cluster
- **Vanlige feil** - Hvilke feil en lett kan komme borti og hvordan en håndterer de

Demonstrasjon og videre arbeid - Først demonstreres hver komponent som installeres for å vise hvilken rolle og nytte den har i oppsettet. Hver del av oppsettet vil så bli oppfølgt av en videre anbefaling om hvordan løsningen kan bli klar for et IT-driftsmiljø. Punktet er bygd opp slik:

- **Docker** - Hvordan Docker brukes i et Kubernetes-oppsett
- **Kubernetes** - En større brukerguide for hvordan Kubernetes opereres, samt hvilke anbefalinger prosjektgruppen vil gi for videreutvikling av oppsettet for bedrifter
- **Prosjektets webapplikasjon** - Demonstrasjon av hvordan prosjektets mikrotjenester fungerer

- **Overvåkning med TICK** - Hvordan TICK-stacken brukes i oppsettet og hvordan komponentene kan benyttes i praksis
- **Sikkerhet** - Et kort overblikk over hvordan sikkerhetsbildet har blitt håndtert i forhold til Kubernetes og organisasjonen NIST sine anbefalinger. I tillegg diskuteres ytterligere faktorer en må notere for sikring av et cluster i produksjon

2 Definisjoner og forkortelser

- Kubernetes
  - Cluster:

    Flere noder som styres av Kubernetes.

  - Pods:

    En pod er et sett med kjørende containere (en eller flere) som skal holdes samlet. Disse blir kjørt på nodene i et cluster.

  - Master:

    Tjener som administrerer Kubernetes-cluster. Denne vil inneholde en scheduler for å velge hvor pods skal kjøres, etcd for å lagre data om clusteret, en API-server for å kunne kontrollere clusteret, samt kontrollører for en del andre funksjoner i Kubernetes.

  - Kubelet:

    Agent som kjører på hver enkelt node i Kubernetes clusteret, denne utfører oppgaver på vegne av Kubernetes-masteren på nodene. Eksempler på oppgaver kan være å sette opp en ny pod eller liknende.

  - Namespace:

    Angis til en gruppe med tjenester en ønsker å ha separert. Et namespace vil være med på å definere navn i Kubernetes sin API og gir muligheten til å gi spesifikke policies og autentisering til tjenestene under namespacet.

- Autoskalering:

  Systemet øker eller senker tildelte ressurser til en tjeneste basert på forskjellige variabler. Dette kan baseres på tid eller ressursbruk.

- HPA:

  Horizontal Pod Autoscaler er en ressurs i Kubernetes som kan brukes for å relativt enkelt sette opp horisontal skalering av pods. Dette vil si at antallet pods økes etter hvert som det blir mer last på dem.

- Side-car:

  I prosjektoppgaven blir dette brukt om tjenester som legges ved i pods under utrulling. En kan for eksempel legge ved overvåkningsverktøy i

pod-en for å holde kontroll på hver enkelt container inni.

- YAML:

  Format for å gjøre info lett leselig for mennesker, blir ofte brukt i konfigurasjonsfiler.

- Token:

  En kode som brukes for å verifisere at en kommuniserer med et godkjent program/enhet, kan brukes som innloggingsinfo.

- CI/CD:

  System for utvikling. Målet er å fort kunne rulle ut nye oppdateringer og nås ved bruk av automatisert testing, og gjerne automatisert utrulling. Som regel vil dette innebære at det rulles ut mange mindre endringer oftere hvor hver enkelt oppdatering testes mot fastsatte krav.

- Policy:

  Et sett med regler

- Debugging:

  Feilsøking

- Metrikk:

  Data hentet fra målinger, blir i rapporten brukt om målinger gjennomført på Pods og noder.

- Bottleneck:

  Når kapasiteten til en applikasjon blir begrenset av en enkeltkomponent

- Flagg:

  Brukes i sammenheng med hvordan en prosess skal fungere. Et flagg er en verdi som fungerer som et "signal" ovenfor hva en prosess skal utføre.

- Hypervisor:

  Programvare som virtualiserer ressursene til en fysisk maskin.

3 Installasjon og oppsett

3.1 Kubernetes Master

Installasjonen er basert på Microsoft sin dokumentasjon om Kubernetes Master (Schott, Lang, Kudrayvtsev & Tylenda, 2018) og Dockers guide for installasjon av Docker CE på Linux(Docker, u.d.).

3.1.1 Før en begynner

- Installer og konfigurer Ubuntu Server 16.04

  Dokumentasjonen gruppen fulgte var ikke verifisert på nyere utgivelser av Ubuntu under prosjektets gjennomførelse. Andre Linux-distribusjoner som er sertifisert til å fungere med Kubernetes kan brukes i stedet for Ubuntu, men gruppen kan ikke garantere at hver komponent i dette testmiljøet vil stemme overens.

- Naviger til "root shell"

  Alle kommandoer kjøres som root om det ikke er spesifisert noe annet. For å operere fra root shell skrives det inn:

  Shell:

```
sudo su
```

- Oppdater Advanced Package Tool (apt)

  Shell:

```
apt update -y && apt upgrade -y
```

- Ved redigering av YAML-filer så brukes det mellomrom og ikke tab for å gjøre innrykk
- Om en opplever feil under prosjektets installasjonsguide så har gruppen opprettet et punkt for "vanlige feil" under dokumentasjonen

3.1.2 Installere Docker

Sett opp Docker sitt repository på maskinen

For å bruke containere trenger man en container-engine, som i prosjektets tilfelle er Docker. Om en har tidligere versjoner av Docker installert må en starte med å avinstallere de gamle komponentene:

Shell:

```
apt remove docker docker-engine docker.io containerd
    runc
```

Videre settes opp Docker sitt repository:

- Installer pakker som lar apt bruke repository-et over HTTPS:

  Shell:

  ```
  apt install apt-transport-https ca-certificates
     curl gnupg-agent software-properties-common
  ```

- Legg til Docker sin offisielle GPG-nøkkel:

  Shell:

  ```
  curl -fsSL "https://
     download.docker.com/linux/ubuntu/gpg" | sudo
     apt-key add -
  ```

  Deretter bekreftes det at en har riktig "fingeravtrykk" ved å søke etter de 8 siste karakterene av fingeravtrykket:

- Legger til repository for docker-pakker på OS-et.

  Shell:

  ```
  add-apt-repository "deb [arch=amd64] https://
     download.docker.com/linux/ubuntu $(
     lsb_release -cs) stable"
  ```

Installere Docker CE

Først må en oppdatere apt-pakkeindeks:

Shell:

```
apt update
```

For å installere den nyeste versjonen av Docker CE gjennomføres følgende:
Shell:

```
apt install docker-ce docker-ce-cli containerd.io
```

En kan deretter sjekke om Docker er installert riktig ved å kjøre et "hello-world"-image:

Shell:

```
docker run hello-world
```

3.1.3 Installere kubeadm

Først laster en ned kubeadm sine binaries:

Shell:

```
curl -s https://
   packages.cloud.google.com/apt/doc/apt-key.gpg |
   apt-key add -

cat <<EOF >/etc/apt/sources.list.d/kubernetes.list

deb http://apt.kubernetes.io/ kubernetes-xenial main


EOF

apt update && apt install -y kubelet kubeadm kubectl
```

Deretter må swap space skrus av, dette gjøres ved å først gå til mappen etc/fstab og fjerne linjen som inneholder swap.img hvis denne er her, for så å kjøre swapoff -a:

Shell:

```
nano /etc/fstab #fjern linjen som inneholder "
   swap.img" om den eksisterer

swapoff -a
```

3.1.4 Start opp masteren

Bestem hvilket subnet du vil sette av til clusterene og servicene og initialiser deretter masteren med kubeadm:

Shell:

```
kubeadm init --pod-network-cidr=10.244.0.0/16 --
   service-cidr=10.96.0.0/12
```

Det er lurt å notere "kubeadm join"-kommandoen og dens opprettede "token". Om den utløper kan en ny genereres ved å skrive "kubeadm token create -print-join-command". Config-filen som inneholder informasjonen legges senere inn på masterens noder om en bruker prosjektets script for å sette opp noder.

Avslutningsvis går en ut av root-miljøet ved å skrive "exit" i terminalen og skriver følgende som en vanlig bruker:

Shell:

```
mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf
   $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Nå kan en bruke "kubectl" til å administrere clusteret.

3.1.5 Verifisere masteren

For å teste om masteren er satt opp riktig kan en prøve å hente ut Kubernetes sine pods (som settes opp automatisk):

Shell:

```
kubectl get pods -n kube-system
```

Videre skal "kubectl cluster-info" kunne gi ut informasjon om Kubernetes-masterens API-server og DNS.

3.1.6 Tillate at pods kjøres ut på masteren

I gruppens oppsett er det visse tjenester som må rulles Linux-noder, og ettersom den eneste Linux-noden som gruppen har i prosjektet er master-maskinen må en åpne opp denne for å kjøre pods. Dette gjøres ved å fjerne det de i Kubernetes kaller Taint fra masteren, og det kan gjøres slik:

Shell:

```
kubectl taint nodes --all
   node-role.kubernetes.io/master-
```

En bør da få en respons som sier "node/master-node untainted"

3.2 Installere Flannel som nettverksløsning

Denne guiden er basert på Statemigration sin installasjonsguide for Flannel (statemigration, u.d.).

3.2.1 Forberedelse

Installasjonen utføres på Kubernetes Master:

Først aktiveres ”bridged IPv4-trafikk” for ”iptables chains”:

Shell:

```
sudo sysctl net.bridge.bridge-nf-call-iptables=1
```

3.2.2 Laste ned og konfigurere Flannel

Shell:

```
wget https://raw.githubusercontent.com/coreos/
   flannel/master/Documentation/kube-flannel.yml
```

Følgende gjenstander må sjekkes i “kube-flannel.yml” under “net-conf.json” (portene er nødvendig for at Flannel skal kunne samarbeide mellom Linux og Windows):

- Cluster subnet er riktig oppgitt under “Network”: (10.244.0.0/16 i prosjektets tilfelle)
- “Type” skal være “host-gw”

Eksempelvis vil filen se slik ut:

```
{
        "Network": "10.244.0.0/16",
        "Backend": {
                "Type": "host-gw"
        }
}
```

Videre må "nodeSelector" videredefineres i den samme filen (kube-flannel.yml) siden "DaemonSets" ikke er støttet på Windows. Dette legges til i punktet under "net-conf.json" som er beskrevet med "kind: Daemonset" og "name: kube-flannel-ds". Bla ned til en finner "nodeSelector" under "spec:" og legg til en ny linje under den allerede definerte "arch"-selectoren. I den nye linjen under nodeSelector skrives: "beta.kubernetes.io/os: linux". Om punktet er satt opp riktig burde det slikt ut:

YAML:

```
spec:
  hostNetwork: true
  nodeSelector:
    beta.kubernetes.io/arch: amd64
    beta.kubernetes.io/os: linux
```

3.2.3 Oppstart av Flannel og validering

Start Flannel med:

Shell:

```
kubectl apply -f kube-flannel.yml
```

Ettersom Flannel-containerne bare fungerer på Linux blir det viktig å forhindre denne fra å rulles ut på Windows-nodene. Dette kan gjøres ved å legge inn en "node-selector" på daemonset-et til kube-flannel-ds-amd64 som vist under:

Shell:

```
kubectl patch ds/kube-flannel-ds-amd64 --patch "$(
   cat node-selector-patch.yml)" -n=kube-system
```

Nå vil en kunne se alle Flannel-poddene ved å skrive:

Shell:

```
kubectl get pods --all-namespaces
```

I tillegg burde Flannel-poddene ha fått NodeSelector til å bare være satt til Linux:

Shell:

```
kubectl get ds -n kube-system
```

Sjekk så at DaemonSet for kube-proxy er satt til ”RollingUpdate”:

Shell:

```
kubectl get ds/kube-proxy -o go-template='{{.
    spec.updateStrategy.type}}{{"\n"}}' --namespace=
    kube-system
```

Så må DaemonSet for kube-proxy oppdateres for å peke mot Linux.

Last ned eller opprett filen ”node-selector-patch.yml” basert på denne GitHub-lenken: ”https://github.com/Microsoft/SDN/blob/master/Kubernetes/flannel/l2bridge/manifests/node-selector-patch.yml ”

Kjør deretter følgende for å ta i bruk endringen fra YAML-fila:

Shell:

```
kubectl patch ds/kube-proxy --patch "$(cat
    node-selector-patch.yml)" -n=kube-system
```

3.3 Installasjon av Helm

Installasjonen utføres basert på Helm sin egen installasjonsdokumentasjon (CNCF, u.d.).

Det neste steget for master-maskinen er installasjonen av Helm. Helm er en ”packet manager” for Kubernetes og gjør det enkelt å rulle ut applikasjoner. Helm består av to deler: Helm-klienten og Tiller (Helm sin server). Tiller kjører inne i Kubernetes-clusteret og håndterer utrulling av applikasjonene. Helm-klienten kjører derimot hvor en ønsker å håndtere administrasjon og utrulling, som gjerne kan være på en laptop eller en del av et ”CI/CD”-system (continuous integration and continuous delivery). Siden prosjektet

gjennomføres på litt mindre skala vil begge delene av Helm kjøres på master-maskinen, der Tiller kjøres som en container i clusteret og Helm installeres lokalt.

3.3.1 Helm-klient

Shell:

```
curl https://raw.githubusercontent.com/helm/helm/
   master/scripts/get > get_helm.sh

chmod 700 get_helm.sh

./get_helm.sh
```

3.3.2 Tiller

Etter Helm-klienten er installert kan en ta i bruk Helm sine kommandoer. For å spinne opp en Tiller-server skrives:

Shell:

```
helm init --node-selectors "beta.kubernetes.io/os"="
   linux"
```

Når Tiller er ferdiginstallert kjøres følgende kommando for å oppgradere Tiller:

Shell:

```
helm init --upgrade
```

Nå mangler det bare en "service account" og "cluster-rolebinding" for at navnerommet tiller kan brukes. Etter opprettelsen må Tiller så oppdateres, skriv følgende kommandoer på masteren:

Shell:

```
kubectl create serviceaccount --namespace
   kube-system tiller

kubectl create clusterrolebinding
   tiller-cluster-rule --clusterrole=cluster-admin
   --serviceaccount=kube-system:tiller

kubectl patch deploy --namespace kube-system
   tiller-deploy -p '{"spec":{"template":{"spec":{"
   serviceAccount":"tiller"}}}}'
```

Nå skal masteren være klar for å rulle ut applikasjoner med Helm, noe som vil tas i bruk for bl.a. TICK-stack.

3.4 MetalLB

Installasjonen er basert på MetalLB sin installasjonsdokumentasjon(MetalLB, u.d.-a).

Den siste komponenten som skal installeres på Kubernetes-masteren er prosjektets lastbalanserings-verktøy. Denne kan installeres direkte ved å skrive:

Shell:

```
kubectl apply -f https://raw.githubusercontent.com/
   google/metallb/v0.7.3/manifests/metallb.yaml
```

Etter kommandoen er kjørt vil MetalLB rulle ut en ”controller” som vil ta for seg utdeling av IP-adresser, samt en ”speaker” som gjør at tjenestene kan nås over riktig protokoll. I YAML-filen er service-account og role-based-permissions i tillegg definert.

Det eneste MetalLB mangler for å være funksjonell nå er et ”ConfigMap”. Her valgte prosjektgruppen å gå for en ”layer 2 configuration” som vil nærmere beskrives i punkt 4.2.7 ”Anbefalt videre arbeid”. For å opprette et slikt configmap så opprettes en YAML-fil med ”kind: ConfigMap” der en definerer det eksterne adresserommet tjenester skal kunne nås med:

YAML:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - [startadresse]-[sluttadresse]
```

Når en setter av et adresserom kan en også bruke CIDR-notasjon for å oppgi et subnet, slik som "192.168.0.0/24". Dette vil i praksis være de adressene MetalLB vil kunne gi servicene på Kubernetes for å kunne nås utenfra. For å sette den opprettede konfigurasjonsfilen i bruk må den rulles ut på Kubernetes:

Shell:

```
kubectl apply -f [configfil]
```

Nå som MetalLB er satt opp vil det begynne å dele ut eksterne IP-adresser til tjenester av typen "LoadBalancer". Dette vil kunne bekreftes senere i driftsrapporten ved å rulle ut prosjektets webapplikasjon.

3.5 Oppsett av Windows-noder

Installasjon og oppsett basert på Microsoft sin guide for å legge til Windows-noder i Kubernetes-cluster(Schott, Scherer & Pengweiqhca, 2018).

3.5.1 Før en begynner

- Installer og konfigurer Windows Server 2019

  Driftsrapporten egnes 1809-utgivelsen av Windows, testing og verifisering av nyere versjoner har ikke blitt gjennomført.

- Planlegg IP-adresser for clusteret

  I prosjektgruppens oppsett vil service-subnet være 10.96.0.0/12, cluster-subnettet vil være 10.244.0.0/16 og Kubernetes sin DNS service-IP vil være 10.96.0.10.

- Skru av anti-spoofing protection

  Når en skal rulle ut Kubernetes på virtuelle maskiner er det viktig at hvert fysiske nettverkskort kan operere med flere forskjellige adresser. Dette ettersom maskinen skal operere på vegne av containerne som opererer med egne IP og MAC-adresser. Av denne grunn er det nødvendig å aktivere addresse-spoofing for VM-ene. Dette kan ordnes ved å aktivere "promiscuous mode" i VMware eller MAC-address-spoofing i Hyper-V.

  I Hyper-V kan dette løses med disse to PowerShell-linjene:

```
Set-VMProcessor -VMName [Name] -
   ExposeVirtualizationExtensions $true

Get-VMNetworkAdapter -VMName [Name] |
   Set-VMNetworkAdapter -MacAddressSpoofing On
```

- Overføre filen /.kub/config fra Linux-masteren til en delt mappe

  I prosjektet løste gruppen dette ved å koble opp en FAT32-disk på masteren, flytte over filen, og deretter flytte disken over på en Windows-node. Derfra ble filen lagt i den delte mappen for Windows-maskiner som prosjektgruppen opprettet på vertsmaskinen.

3.5.2 Script for installasjon av Windows-node

Prosjektgruppen ønsker å vise to metoder for å installere Windows-nodene: en ved hjelp av PowerShell-script på noden, og en manuell metode. Scriptene er navngitt "Installer_Docker.ps1", "Installer_Kubernetes_Flannel.ps1" og "Installer_Telegraf.ps1" og er oppført som vedlegg i bacheloroppgaven.

For å installere med script er det et par ting som må ordnes manuelt:

- En virtuell maskin må opprettes med ferdiginstallert OS (Windows Server 2019 eller Windows Server 2016 versjon 1809)
- Maskinen må være koblet på nett, og tildelt en fast IP

- Kubernetes og Telegrafs config-filer må ligge tilgjengelige på oppgitt lagringsplassering for noden

Prosjektgruppen har valgt å dele installasjonsscriptet opp i tre forskjellige deler. En del for installasjon av Docker, en for installasjon av Telegraf og en del for installasjon av Kubernetes, Flannel og oppkobling til clusteret. Det er her viktig at scriptet for Docker-installasjon kjøres før Kubernetes, men telegraf-installasjonen kan skje uavhengig av de to andre.

Script for Kubernetes- og Telegraf-installasjon krever at config-filer ligger tilgjengelig på nettverksdeltlagring som blir oppgitt som variabel til scriptet (standard er "\\win-host\nettverksdelt").

3.5.3 Klargjøre Windows node manuelt

Et frivillig første-punkt vil være å gi maskinen et lett gjenkjennelig navn.

Deretter må Kubernetes sin container-engine installeres. Kubernetes bruker Docker som sin container-engine og denne installeres på følgende måte:

PowerShell:

```
Install-Module -Name DockerMsftProvider -Repository
   PSGallery -Force

Install-Package -Name Docker -ProviderName
   DockerMsftProvider

Restart-Computer -Force
```

Når maskinen har startet opp igjen kjøres denne linjen for å starte Docker:

PowerShell:

```
Start-Service Docker
```

3.5.4 Opprette Kubernetes pause-image

Kubernetes trenger et såkalt pause-image som brukes for å vite når en mister tilgangen til pods/noder. Denne er nødvendig for at tjenester skal kunne rulles ut på noden, og opprettes ved å kjøre følgende kommandoer:

PowerShell:

```
docker pull mcr.microsoft.com/k8s/core/pause:1.0.0

docker tag mcr.microsoft.com/k8s/core/pause:1.0.0
   kubeletwin/pause
```

3.5.5 Hente konfigurasjons- og Kubernetes-filer

Begynn ved å opprette en mappe for Kubernetes sine Binary-filer, utrullingsscript og config-filer.

PowerShell:

```
Mkdir c:\k
```

Hent så sertifikatfilen fra " /.kube/config" fra Kubernetes Master-maskinen over til den nyopprettede "C:\k"-mappen.

Deretter kan en laste ned Kubernetes sine Binary-filer. Noden trenger "kubectl", "kubelet" og "kube-proxy" sine binaries. Disse kan finnes i "CHANGELOG.md"-filen fra Kubernetes sin nyeste utgivelse: "https://github.com/kubernetes/kubernetes/releases/"

3.5.6 Legge noden inn i et Flannel-cluster

I gruppens oppsett brukes Flannel til å håndtere nettverkstrafikk for Kubernetes. Med Flannel kan Kubernetes gå ut ifra at hver pod har sin egen unike IP-adresse inne i clusteret selv om hostmaskinen kun har en IP-adresse.

Først må en hente exe-fil for Flannel. Denne kan hentes her: "https://github.com/coreos/flannel/releases/" og legges i "c:\k\" på noden. Etter dette trengs installasjonsfilen laget av Microsoft for å sette opp Flannel, og melde den inn i clusteret. Denne kan finnes her: "https://github.com/Microsoft/SDN/tree/master/Kubernetes/flannel" under navnet "start.ps1". Også denne filen legges på "c:\k\".

For å fullføre installasjonen av Flannel og melde den inn i clusteret kjøres følgende kommandoer:

PowerShell:

```
Cd \k

Chcp 65001

start.ps1 -ManagementIP [Node-IP] -NetworkMode "
   l2bridge" -LogDir "c:\k\logs"
```

Når dette er gjort vil det ha åpnet seg tre forskjellige PowerShell vinduer, ett for kubelet, ett for kube-proxy og ett for flanneld.

3.5.7 Sette opp Kubernetes-filer som Windows-service

I prosjektgruppens oppsett har de tre tjenestene Kubernetes benytter på noden blitt satt opp som Windows-services, det vil si at de startes opp som bakgrunnstjenester ved oppstart av noden. For å sette opp dette tar prosjektgruppen i bruk verktøyet NSSM (The Non Sucking Service Manager) som kan hentes her: "https://nssm.cc/download"

For å sette opp de tre tjenestene slik det ble gjort i prosjektet kan en laste ned filen register-svc.ps1 fra Microsofts GitHub (https://github.com/Microsoft/SDN/tree/master/Kubernetes/flannel), og gjøre en liten endring på denne. På linje 25 i denne (Begynner med ".\nssm.exe set $KubeletSvc AppParameters") legger en til følgende parameter på slutten av linja: " --read-only-port 10255". Dette vil tillate at data om noden/clusteret kan hentes ut uten at det kreves autentisering.

Etter dette kan scriptet kjøres med nødvendige variabler, og ble i prosjektet kjørt slik:

PowerShell:

```
register-svc.ps1 -NetworkMode "l2bridge" -
   ManagementIP [Node-IP] -LogDir "c:\k\logs\"
```

Når dette scriptet er fullført skal en kunne starte maskinen på nytt og se at kubelet, kube-proxy og flanneld dukker opp blant service-ene i Windows. Dette kan testes ved å kjøre "Get-Service" i Powershell.

3.6 Utrulling av prosjektets webapplikasjon

Gruppen har i løpet av oppgaven utviklet et enkelt mikrotjeneste-oppsett bestående av tre forskjellige pods. Denne applikasjonen kan benyttes for å demonstrere flere nyttige egenskaper ved Kubernetes som autoskalering og lastbalansering.

For å rulle ut applikasjonen har gruppen satt sammen tre forskjellige YAML-filer, for å legge disse inn i clusteret kan alle tre samles i en enkelt mappe og kjøre følgende kommando:

PowerShell:

```
kubectl apply -f [mappenavn]
```

De tre YAML-filene kan finnes som vedlegg til dette dokumentet og er navngitt win-webapp.yml, win-mikrotjeneste-1.yml og win-mikrotjeneste-2.yml

Når dette er gjort skal en få opp tre services, tre deployments, tre HPA-er og en til fire pods per deployment. Dette kan en sjekke ved følgende kommando:

PowerShell:

```
kubectl get svc,deployment,hpa,pods -n app
```

```
kub@Master-Node:~/skalering$ kubectl get svc,deployment,hpa,pods -n app
NAME                            TYPE           CLUSTER-IP       EXTERNAL-IP    PORT(S)        AGE
service/win-mikrotjeneste-1     NodePort       10.110.100.41    <none>         80:31594/TCP   20d
service/win-mikrotjeneste-2     NodePort       10.107.159.234   <none>         80:32746/TCP   20d
service/win-webapp              LoadBalancer   10.96.178.37     129.241.99.5   80:32659/TCP   6d21h

NAME                                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.extensions/win-mikrotjeneste-1  3/4     4            3           20d
deployment.extensions/win-mikrotjeneste-2  1/1     1            1           20d
deployment.extensions/win-webapp           3/3     3            3           6d21h

NAME                                                          REFERENCE                          TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
horizontalpodautoscaler.autoscaling/win-mikrotjeneste-1       Deployment/win-mikrotjeneste-1     77%/50%   1         4         4          128m
horizontalpodautoscaler.autoscaling/win-mikrotjeneste-2       Deployment/win-mikrotjeneste-2     1%/50%    1         4         1          128m
horizontalpodautoscaler.autoscaling/win-webapp                Deployment/win-webapp              41%/50%   1         4         3          81m

NAME                                    READY   STATUS              RESTARTS   AGE
pod/win-mikrotjeneste-1-5b666f56b4-7b789   1/1     Running             0          38m
pod/win-mikrotjeneste-1-5b666f56b4-jcb6v   1/1     Running             0          38m
pod/win-mikrotjeneste-1-5b666f56b4-vhj2b   0/1     ContainerCreating   0          38m
pod/win-mikrotjeneste-1-5b666f56b4-wvsst   1/1     Running             0          93m
pod/win-mikrotjeneste-2-58669b5854-tlsqp   1/1     Running             0          152m
pod/win-webapp-6c6bc79bd6-4bbpl            1/1     Running             0          53m
pod/win-webapp-6c6bc79bd6-c7qlp            1/1     Running             0          93m
pod/win-webapp-6c6bc79bd6-rx7xx            1/1     Running             0          62m
```

Figur 1: Komponentene i webappen

3.7 Autoskalering av tjenester

Oppsett basert på Kubernetes sin dokumentasjon om HPA(Kubernetes, 2019c).

"Horizontal Pod Autoscaler" er Kubernetes sin programvare for automatisk skalering. For å implementere dette på sitt eget cluster må en først rulle ut en "metrics server". Denne samler metrikk fra pods på clusteret med API-kall. Etter dette kan en så konfigurere hvordan skaleringen skal utføres på de forskjellige deployment-ene i clusteret.

3.7.1 Utrulling av Metrics Server

For å rulle ut en metrics server må en laste ned filene dens til Kubernetes-master:

Shell:

```
git clone https://github.com/kubernetes-incubator/
   metrics-server.git
```

Siden prosjektets oppsett har to Windows-noder og en Linux master vil det være nødvendig å spesifisere hvilken maskin metrics-server skal kjøre på. Dette utføres ved å legge ved en "nodeSelector" i YAML-filen til metrics-server.

Naviger til "/metrics-server/deploy/1.8+" og endre filen "metrics-server-deployment.yaml" med ønsket verktøy. "nodeSelector" plasseres under "spec" på følgende måte: NB: Pass på at det brukes mellomrom ved innrykk og ikke tab. Dette nevnes også under punkt 3.9 "Vanlige feil".

YAML:

```
spec:
  nodeSelector:
    beta.kubernetes.io/os: linux
```

Deretter må en navigere seg inn i mappen "metrics-server" og skrive følgende:

```
kubectl create -f deploy/1.8+/
```

Sjekk om tjenesten kjører ved å skrive ”kubectl get pods -n kube-system”

3.7.2 Konfigurere HPA

For å kunne benytte HPA må tjenesten på clusteret allerede ha to spesifikasjoner i deployment-filen sin: ”limit” og ”request”. Dette spesifiseres under ”spec” i YAML-filer av typen ”Deployment”. For hver av mikrotjenestene i webapplikasjonen ble ressursbruken spesifisert på denne måten:

NB: Om en benytter gruppens webapplikasjon vil ikke stegene under være nødvendige å utføre. Gruppen har allerede satt opp denne applikasjonen til å fungere ved å fylle inn resource-limit og requests.

YAML:

```
spec:
  template:
    spec:
      containers:
        resources:
          requests:
            cpu: "50m"
          limits:
            cpu: "200m"
```

”m” i CPU-spesifikasjonene referer til ”millicores”. I dette tilfellet krever tjenesten 50 millicores og har en grense på 200 millicores. HPA støtter også skalering basert på andre grunnlag. Om en skal skalere basert på f.eks. minnebruk og CPU spesifiseres det på samme måte:

YAML:

```yaml
spec:
  template:
    spec:
      containers:
        resources:
          requests:
            cpu: "50m"
            memory: "512Mi"
          limits:
            cpu: "200m"
            memory: "256Mi"
```

For å videre sette i gang HPA så må en opprette en "autoscale"-instans for tjenesten med kubectl. Her bestemmes hvilke mål Kubernetes skal oppnå, samt hvor mange instanser den skal kunne opprette. Eksempelvis kan autoskalering aktiveres slik:

Shell:

```
kubectl autoscale deployment [tjeneste] --
   cpu-percent=50 --min=1 --max=10
```

Her vil Kubernetes prøve å bruke 50% av tjenesten sin CPU-limit, der den må ha minst en instans og har en grense på ti. Dette kan også defineres i ei YAML-fil ved å lage et objekt av typen "HorizontalPodAutoscaler" som vist under:

YAML:

```yaml
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: win-webapp
  namespace: app
  labels:
    app: win-webapp
spec:
  scaleTargetRef:
    apiVersion: extensions/v1beta1
    kind: Deployment
    name: win-webapp
  minReplicas: 1
  maxReplicas: 4
  metrics:
  - type: Resource
    resource:
    name: cpu
    target:
      type: Utilization
      averageUtilization: 50
```

3.8 TICK-stack

Installasjonsguiden er basert på InfluxData sin egen dokumentasjon(Zampolin, 2016).

3.8.1 Nedlasting og installasjon av TICK-charts

Først og fremst må repository-ene for TICK-kompenentene lastes ned fra GitHub:

Shell:

```
git clone https://github.com/influxdata/tick-charts.
   git
```

Endrer chronograf/values.yaml til å oppgi service-type som NodePort:

YAML:

```
service:
  type: NodePort
```

Det må også legges inn en Node-selector på alle tjenestene som bare fungerer på Linux (Chronograf, Kapacitor, Telegraf-s og InfluxDB). Dette gjøres som demonstrert under med InfluxDB sin values-fil.

YAML:

```
nodeSelector: {
        "beta.kubernetes.io/os":"linux"
}
```

For å få telegraf-s til å fungere riktig må en inn i values fila til denne tjenesten og legge til følgende på inputs:

YAML:

```
config:
  Inputs:
  - cpu:
      percpu: false
      totalcpu: true
```

For at telegraf-ds skal kunne hente info fra Kubernetes må en liten endring i values-filen utføres. Verdien for kubernetes-url må endres til masterens IP-adresse:

YAML:

```
config:
  inputs:
  - kubernetes:
      url: "http://192.168.137.50:10255"
```

Deretter må det gjøres en endring i config-fila for kubelet på maskinen for å gjøre infoen tilgjengelig. Endringen består av å legge til "--read-only-port 10255" sist på siste linje i filen plassert på "/etc/systemd/system/kubelet.service.d/10-kubeadm.conf" på masteren:

Conf:

```
ExecStart=/usr/bin/kubelet $KUBELET_KUBECONFIG_ARGS
   $KUBELET_CONFIG_ARGS $KUBELET_KUBEADM_ARGS
   $KUBELET_EXTRA_ARGS -$ARGS --read-only-port 10255
```

InfluxData har lagt opp til at en kan bruke Helm for å rulle ut TICK-stacken, så videre navigerer en inn i den nyopprettede mappen "tick-charts". Herfra kan hver del av TICK installeres med Helm:

Shell:

```
helm install --name data --namespace tick ./
   influxdb/

helm install --name polling --namespace tick ./
   telegraf-s/

helm install --name hosts --namespace tick ./
   telegraf-ds/

helm install --name alerts --namespace tick ./
   kapacitor/

helm install --name dash --namespace tick ./
   chronograf/
```

Etter hver komponent er ferdiginstallert kan en hente ut Chronograf som vil fungere som brukergrensesnittet for stacken:

Shell:

```
kubectl get svc -n tick -l app=dash-chronograf
```

Gå deretter inn i Chronograf i nettleseren ved å skrive IP-adressen og porten dens. Når siden har lastet må InfluxDB og Kapacitor konfigureres:

- InfluxDB URL: "http://data-influxdb.tick:8086"
- Kapacitor URL: "http://alerts-kapacitor.tick:9092"

Både InfluxDB og Kapacitor vil i tillegg kreve et brukernavn og passord som en selv velger.

3.9 Vanlige feil

3.9.1 Kubectl "Connection was refused"

```
kub@lnxmaster:~$ kubectl get node
The connection to the server 192.168.137.151:6443 was refused - did you specify
the right host or port?
```

Figur 2: Nektet tilkobling til API-server

Dette kan vanligvis løses med fire trinn:

1. Logg på masteren som superuser: "sudo su"
2. Kjør: "swapoff -a"
3. Returner tilbake til innlogget bruker
4. Åpne opp for bruk av kubectl: "strace -eopenat kubectl cluster-info"

3.9.2 Internal-IP for node er ikke i clusteret

```
kub@Master-Node:~$ kubectl get pods -o wide
NAME                                READY   STATUS              RESTARTS   AGE
win-webserver-7f64f4ff9f-2d5g5      0/1     ContainerCreating   0          <inva
win-webserver-7f64f4ff9f-tj5xp      0/1     ImagePullBackOff    0          <inva
kub@Master-Node:~$ kubectl get node -o wide
NAME          STATUS   ROLES    AGE   VERSION   INTERNAL-IP      EXTERNAL-IP
master-node   Ready    master   24h   v1.13.4   192.168.137.50   <none>
win-node-01   Ready    <none>   12m   v1.13.4   10.244.2.2       <none>
```

Figur 3: Nodens interne IP er utenfor cluster

Dette kommer mest sannsynlig av at "MAC address spoofing" ikke er aktivert. Resultatet av dette er at en ikke kan rulle ut applikasjoner på clusteret. Hvis en bruker Hyper-V dette kan i PowerShell på Host-maskinen gjøres med kommandoen:

PowerShell:

```
Get-VMNetworkAdapter -VMName [Name] |
   Set-VMNetworkAdapter -MacAddressSpoofing On
```

Når dette er gjort kan det være lurt å starte noden på nytt. Om pod-ene enda ikke kan rulles ut kan det være lurt å slette dem og la Kubernetes rulles ut nye instanser.

3.9.3 Feil med YAML-fil

```
kub@Master-Node:~$ kubectl apply -f win-web-f.yml
service/win-web-f created
error: error parsing win-web-f.yml: error converting YAML to JSON: yaml: line 12
: found character that cannot start any token
```

Figur 4: Klarte ikke konvertere YAML til JSON

Ofte kan bruk av tab ved innrykk i .yml-filer gi feilmeldinger om at filen ikke er gyldig. Ved slike tilfeller vil det være nødvendig å gå gjennom endringene en har gjort i filen og sjekke om en har husket å bruke mellomrom for innrykk. Kan også komme av andre feil ved strukturen til filen, som feil antall innrykk eller lignende.

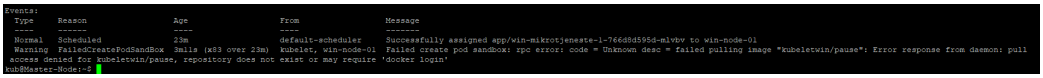
3.9.4 Pods når ikke DNS/Service-IP

Dette skyldes mest sannsynlig feiloppsatt Flannel. Gå til Flannel-konfigurasjonsfilen som ligger på "C:\k\cni\config\cni.conf" og sørg for at adresserommene stemmer med clusteret. I prosjektets tilfelle er det følgende policies som gjelder:

```
"policies":  [
                {
                    "Name":  "EndpointPolicy",
                    "Value":  {
                                  "Type":  "OutBoundNAT",
                                  "ExceptionList":  [
                                                        "10.244.0.0/16",
                                                        "10.96.0.0/12",
                                                        "192.168.137.0/24"
                                                    ]
                              }
                },
                {
                    "Name":  "EndpointPolicy",
                    "Value":  {
                                  "Type":  "ROUTE",
                                  "DestinationPrefix":  "10.96.0.0/12",
                                  "NeedEncap":  true
                              }
                },
                {
                    "Name":  "EndpointPolicy",
                    "Value":  {
                                  "Type":  "ROUTE",
                                  "DestinationPrefix":  "192.168.137.51/32",
                                  "NeedEncap":  true
                              }
                }
            ]
        }
}
```


Figur 5: Eksempeloppsett for flannel-policies fra cni.conf

3.9.5 "Failed pulling image kubeletwin/pause"



Figur 6: Klarte ikke hente image "kubeletwin/pause"

Dette betyr at pause-image som skal være en del av hver enkelt pod ikke er tilgjengelig på noden. For å fikse dette kjøres følgende to kommandoer fra powershell på noden:

PowerShell:

```
docker pull mcr.microsoft.com/k8s/core/pause:1.0.0

docker tag mcr.microsoft.com/k8s/core/pause:1.0.0
   kubeletwin/pause
```

4 Demonstrasjon og videre anbefalinger

I denne seksjonen av rapporten vil prosjektgruppen demonstrere og diskutere hvordan en i praksis vil kunne bruke et lokalt Kubernetes-oppsett som et driftsverktøy for tjenestelevering. I dette avsnittet vil gruppen presentere vanlige bruksområder for hver delkomponent i systemet, samt et overordnet bilde over hvilket arbeid som en bør utføre for å gjøre systemet produksjonsverdig.

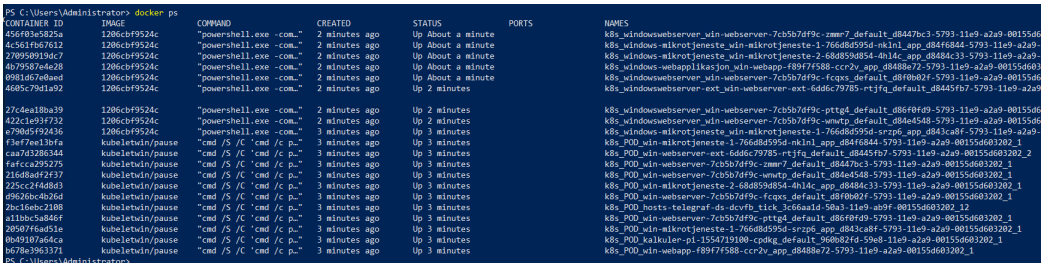
4.1 Docker

I et driftsmiljø basert i Kubernetes vil en ikke i utgangspunktet måtte detaljstyre Docker på et administrativt nivå. Docker sin rolle er likevel kritisk siden det er denne komponenten som faktisk virtualiserer operativsystemet til noden for å kjøre containere. I et Kubernetes-driftsmiljø vil en i hovedsak ha tre oppgaver som involverer Docker:

- Opprette Docker-images for utrulling
- Sjekke hvilke containere som faktisk kjører på noden
- Oppgradere til en nyere Docker-versjon

I dette prosjektet har det naturligvis vært et langt større driftsfokus enn hos utvikling, dermed har gruppen lite å presentere i form av hvordan en pakker en applikasjon til å kunne kjøres som container. I et reelt miljø vil en som regel ha dedikerte utviklingsgrupper som står for opprettelsen av applikasjonen og for å gjøre den klar for Docker.

Under debugging vil det av og til være lurt å ta seg en tur inn på arbeidsnoden og se om Docker fungerer som det skal. Om en har kommet borti større feil kan informasjonen en får tilsendt til Kubernetes-masteren ikke samsvare med hva noden som kjører Docker faktisk gjør. I prosjektets tilfelle har en i noen tilfeller gått inn og verifisert at noden kjører de containerne den skal. På gruppens Windows-noder får en denne outputen når en skrev inn ”docker ps” (Docker Processess) i et PowerShell-vindu:



Figur 7: Containere kjørende på windows-node

Her kan en bekrefte at hver tjeneste kjører og har et eget "/pause"-image. Dette imaget inneholder navnerommet for nettverket til containeren og gjør i praksis veldig lite. Poenget med dette imaget er at om en container fjernes og rulles ut igjen så har den allerede på plass nettverksinnstillingene sine.

Til slutt vil en gjerne kunne oppgradere og verifisere Docker-versjonen til arbeidsnoden. Dette har en viktig rolle i samspillet med Kubernetes. På Kubernetes-versjonen prosjektgruppen hadde (v1.14) så var disse Docker-versjonene støttet: 1.13.1, 17.03, 17.06, 17.09, 18.06, 18.09. Under gjennomførselen av prosjektet kjørte gruppen versjon 18.09.3 (dette sjekkes med å skrive "docker version" på noden). Når en omsider må oppgradere Docker-versjonen skrives dette på Windows-noden det gjelder:

PowerShell:

```
Install-Package -Name Docker -ProviderName
   DockerMsftProvider -Verbose -Update
```

4.2 Kubernetes

Det er en rekke tjenester Kubernetes ordner av seg selv ved installasjonen. Blant disse er:

- Intern-nettet til clusteret (cluster-ip og service-ip)
- DNS (core-dns)
- API-kommunikasjon mellom sine tjenester (kube-apiserver)
- Lastbalansering mot pods (kube-proxy)

Det en vanlig IT-administrator vil kunne gjøre i Kubernetes er stort sett konfigurasjon og administrering av utrullinger, i tillegg til oppdateringer av noder. Det fleste av disse oppgavene utføres i Kubernetes sin kommandolinje, noe som vil gjennomgås i de følgende punktene. Videre vil gruppen diskutere hvilket arbeid som bør gjøres utover hva som har blitt utført i prosjektet for å gjøre gruppens oppsett mer egnet for reell IT-drift.

4.2.1 Kubectl

Kubectl er kommandolinjen til Kubernetes som skriver mot API-serveren til clusteret verktøyet opererer. Oppbyggelsen av kubectl-kommandoer er som følge: kubectl [command] [Type] [Name] [Flags](Kubernetes, 2019d).

- **Command** - spesifiserer operasjonen en vil utføre, f.eks. create, get, describe og delete
- **Type** - spesifiserer ressurstypen en vil behandle, f.eks. pods, deployment og service
- **Name** - spesifiserer navnet til ressursen, f.eks. navnet på en pod
- **Flags** - spesifiserer valgfrie kategorier, f.eks. om en bare vil behandle ressurser på et namespace

Under listes en del av kubectl sine kommandoer som prosjektet har benyttet underveis:

Forklaring av Kubernetes sine ressurser, f.eks. deployment eller pods:

```
kubectl explain [type]
```

Utrulling av en tjeneste/konfigurasjon på Kubernetes sitt cluster:

```
kubectl apply -f [fil]
```

Utlisting av objekter av en viss ressurs i et gitt namespace, eksempler på ressurser er service og node. Eksempler på namespace er default og kube-system:

```
kubectl get [type] -n [namespace]
```

”Describe” kan beskrive enkelt-objekter og gi info om hvilken status disse har. Et eksempel kan være å hente informasjon om en enkelt pod med ”kubectl describe pod etcd-master-node -n kube-system”:

```
kubectl describe [type] [name] -n [namespace]
```

”Delete” sletter et objekt fra clusteret:

```
kubectl delete [type] [name] -n [namespace]
```

”Drain” fjerner alle pods som kjører på en gitt node, og hindrer nye pods i å rulles ut på denne:

```
kubectl drain [nodenavn]
```

”Scale” kan skalere en utrulling til et gitt antall pods:

```
kubectl scale deployment [name] --replicas=[antall]
```

”Version” skriver ut versjonen av kubernetes:

```
kubectl version
```

Oppgraderer Kubernetes på Linux:

Shell:

```
sudo apt-update && upgrade

sudo kubeadm upgrade apply [versjonnummer]
```

4.2.2 Oppdatering av applikasjoner

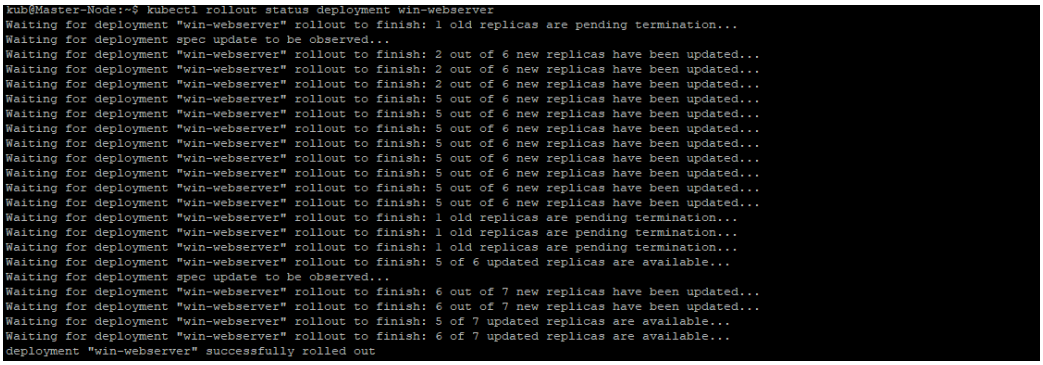
For å oppdatere tjenestene en har på clusteret til en nyere versjon så må bare deployment-en endres. Dette betyr at tjenesten vil fortsatt nås på samme måte, ettersom dette bestemmes av service. For å oppdatere en deployment så ruller en rett og slett bare YAML-filen ut på nytt:

Shell:

```
kubectl apply -f [YAML-fil]
```

En kan videre se Kubernetes sin fremgang i utrullingen ved å skrive:

```
kubectl rollout status deployment [tjenestenavn]
```



Figur 8: Status for utrulling av en tjeneste

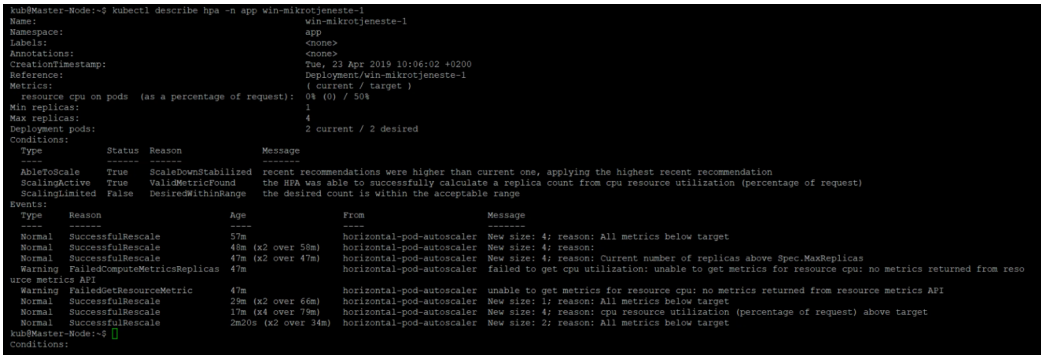
4.2.3 Automatisk skalering av applikasjoner

For å få en oversikt over en tjeneste sin autoskalering kan en skrive:

Shell:

```
kubectl describe hpa -n [namespace]
```

Om en så kjører en test der en belaster tjenesten sin vil en kunne se loggen for hva som ble utført og hvorfor. Eksempelvis kan en se at gruppens "win-mikrotjeneste-1" skaleres automatisk av Kubernetes:



Figur 9: Skaleringshendelser fra HPA

En faktor en burde merke seg er at HPA venter fem minutter før den skalerer ned og tre min før den kan oppskalere etter at det har skjedd en skalering. For å gjøre autoskaleringen mer eller mindre aggressiv så må det endres i "kube-controller-manager". Komponentens YAML-fil ligger under "/etc/kubernetes/manifests" og har en rekke "flagg" en kan endre for å gjøre HPA bedre tilpasset sitt eget Kubernetes-cluster:

1. Endre HPA sitt intervall for uthenting av data (30 sekunder som standard):

   Dette kan endres med å legge til "horizontal-pod-autoscaler-sync-period" sammen med ønsket verdi.

2. Endre HPA sin ventetid for å skalere tjenester opp (default 3 minutt):

   Legg til "horizontal-pod-autoscaler-upscale-delay" med ønsket verdi.

3. Endre HPA sin ventetid for å skalere tjenester ned (default 5 minutt):

   Legg til "horizontal-pod-autoscaler-downscale-delay" med ønsket verdi.

HPA kan videre konfigureres til å skalere på egendefinert metrikk som hentes fra Kubelet på noden(Kubernetes, 2019b). Dette nevnes siden det ofte vil være hensiktsmessige grunnlag for å skalere basert på f.eks. innkommende nettverkstrafikk. I testmiljøet forsøkte gruppen å oppnå dette med autoskalering i Kapacitor, som er en del av overvåkingssystemet TICK. Denne løsningen ville i utgangspunktet også kunne tilbydd egendefinert skalering. Grunnene til at dette ikke ble implementert blir diskutert i punkt 4.2.7 "Anbefalt videre arbeid". HPA ble dermed bare testet og konfigurert med CPU- og minneskalering siden det var backupløsningen for testmiljøet.

4.2.4 Utrulling av tjenester - YAML og HELM

YAML-filer

Om en ser på YAML-filen til gruppens webapplikasjon så kan en se at filen er oppbygd av forskjellige ressurser definert med "kind:". I webapplikasjonen definerte gruppen:

- **Service** - Bestemmer hvordan applikasjonen skal gjøres tilgjengelig.
- **Deployment** - Hvilket image skal hentes, hvilken kode skal kjøres, hvilken node den skal rulles ut på og hvilket namespace den skal ligge under.
- **HorizontalPodAutoscaler** – Oppretter parameter for autoskalering slik brukeren ikke må utføre dette manuelt
- **NetworkPolicy** – Hvilke nettverksregler Flannel skal administrere for utrullingen, gjerne regler for inn- og utdata

Parameterne over er noen få eksempler av Kubernetes sin "API-resource". For å få en fullstendig liste over alle ressursene kan en skrive "kubectl get api-resource". Hver av disse kan konfigureres i YAML-filer og rulles ut på clusteret med "kubectl apply -f [YAML-fil]". Noe som videre vil nevnes i punkt 4.2.6 "Best practice" er at en gjerne vil samle utrullingen av slike ressurser i samme YAML-fil når det er hensiktsmessig. Dette vil gjøre utrulling av applikasjoner mer oversiktlige og bedre organisert.

Om en skal rulle ut sin egen applikasjon på et Kubernetes-cluster vil det være nødvendig å sette seg inn i hvilke API-ressurser en kommer til å få bruk for. En lur måte å holde selve applikasjonen separat fra konfigurasjonen en gjør i Kubernetes er å ta i bruk "ConfigMap". Et ConfigMap kan binde konfigurasjonsfiler, kommandolinjeargument, portnummer o.l. til pods sine containere. På denne måten kan en slippe å ha forskjellige utgaver av applikasjoner for hvert cluster en skal kjøre applikasjonen i.

For å rulle ut en YAML-fil på et Kubernetes-cluster skrives:

Shell:

```
kubectl apply -f [YAML-fil]
```

Helm

Helm er som nevnt i installasjonen en "pakkeadministrator" for Kubernetes. Dette oppnår programvaren med å pakke Kubernetes-ressurser i et format som de kaller for "Helm charts". En chart kan brukes til å rulle ut alt fra enkle applikasjoner til fulle webstacker som inkluderer databaser, cacher, webservere og lignende. I gruppens testmiljø ble Helm brukt til å rulle ut TICK-stacken som InfluxData har samlet i en Helm chart kalt "tick-charts".

Helm har i tillegg sin egen kommandolinje som installeres lokalt på maskinen. Om en har fulgt installasjonsguiden til Helm i prosjektet kan en skrive "helm list" på master-maskinen for å få en oversikt over Helm sine utrullede applikasjoner. Hver chart krever et versjonsnummer, noe som hjelper Helm med å tilby administrasjon av tilbakerulling ved hjelp av kommandoen "helm rollback [utrulling] [versjonsnummer]".

For å rulle ut en Helm-utgivelse på sitt eget cluster skrives "helm install [mappe]". I tillegg kan parameter som f.eks. navn og namespace defineres med flagg i kommandoen: "helm install --name [navn] --namespace [namespace] [mappe med applikasjonen]". For å benytte verktøyet for testing kan man blant annet kjøre "helm install [chart] --dry-run --debug" for å se at hver output er riktig.

For IT-aktører vil Helm kunne være et nyttig rammeverk under utvikling, testing og utrulling av applikasjoner. Verktøyet vil holde applikasjonen (charts) separat fra konfigurasjonen og utrullingen, som er en kombinasjon av en chart og dens konfigurasjon. I tillegg kommer effektiv versjonskontroll og rask utrulling på brukere sine cluster. På bakgrunn av dette vil gruppen anbefale at en i et evt. videre arbeid vil følge med på Helm sin videre utvikling og hvordan verktøyet kan benyttes i egen organisasjon.

4.2.5 Lastbalansering med MetalLB

```
kub@Master-Node:~$ kubectl get svc -n app -o wide
NAME                   TYPE           CLUSTER-IP       EXTERNAL-IP    PORT(S)        AGE     SELECTOR
win-mikrotjeneste-1    NodePort       10.110.100.41    <none>         80:31594/TCP   20d     app=win-mikrotjeneste-1
win-mikrotjeneste-2    NodePort       10.107.159.234   <none>         80:32746/TCP   20d     app=win-mikrotjeneste-2
win-webapp             LoadBalancer   10.96.178.37     129.241.99.5   80:32659/TCP   6d21h   app=win-webapp
kub@Master-Node:~$
```

Figur 10: webapp med ekstern-IP

Siden tjenesten "win-webapp" er av typen "LoadBalancer" så har den automatisk fått tildelt en ekstern IP-adresse. Adressene som MetalLB har tilgjengelig defineres i konfigurasjonsfilen dens, som ble demonstrert i punkt 3.4. MetalLB vil automatisk tildele og fjerne IP-adresser til servicene på clusteret.

MetalLB delegerer videre en av nodene til å ha "eierskap" over IP-adressene, som den videre annonserer ut mot det eksterne nettet. I praksis betyr dette at maskinen sier til ruteren at den eier disse IP-adressene og ruter så videre trafikk til servicene på clusteret. I prosjektetes cluster er det Kubernetes Master som er satt opp til å utføre dette.

4.2.6 Best practice

Kubernetes har publisert egen dokumentasjon om hvilke råd de vil gi til brukere av verktøyet deres(Kubernetes, 2019a). Disse tipsene kan være sentrale for å ha riktig arbeidsflyt og organisering. Dette var noe prosjektgruppen fikk oppleve nytten av underveis.

- Behold konfigurasjonsfiler i en versjonskontroll-mappe før instansen rulles ut på clusteret

  I produksjon vil en helst ha muligheten til å raskt rulle tilbake til forrige stabile versjon av en tjeneste om den nye versjonen viser seg å være trøblete. "ConfigMaps" kan rulles ut fortløpende og kan definere variabler som namespace og data (f.eks. kommandoer og variabler). Ved å ha slike filer organiserte og tilgjengelige kan både produksjon og testmiljø bli mer effektive. Dette viste seg å være svært nyttig ved testing av gruppens webapplikasjon.

- Grupper objekter i en og samme fil når det er mulig

  Om en ser i en typisk deployment-fil så er det ofte ikke bare definert deployment, men også service (hvordan tjenesten skal nås), ReplicaSet (hvor mange pods skal alltid være kjørende) og diverse "role-bindings" til clusteret. Alle disse konfigurasjonene kan i teorien være separerte og rulles ut hver for seg, men vil kunne skape trøbbel i lengden om brukerne ikke er svært rutinerte og har god organisering. Om en tar en titt på prosjektets YAML-fil for webapplikasjonen kan en se disse variablene bli definert.

- Kubectl-kommandoer kan brukes på mapper

  Om en f.eks. har en mappe med konfigurasjonsfiler eller pods som logisk alltid vil rulles ut sammen så kan det oppnås ved å kjøre "kubectl apply -f [mappen]". Kommandoen vil da rulle ut hver YAML-fil i den definerte mappen. For å igjen ta eksempel fra prosjektets webapplikasjon, så besto hele "stacken" av 3 forskjellige YAML-filer: en webapp og to instanser av enkle webservere som gir forskjellig output. I stedet

for å måtte manuelt rulle ut tjenestene hver for seg, så kunne gruppen heller rulle ut alle samtidig fra samme mappe.

- Ta i bruk labels for oversiktlighet

  Utrullinger kan få logiske navn som kan hjelpe bedrifter å gruppere lignende tjenester. Eksempelvis kan det se slik ut i en konfigurasjonsfil:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  namespace: app
  labels:
    app: win-webapp
    tier: frontend
    phase: test
    deployment: v3
  name: win-webapp
```

  Med denne infoen kan administratorene behandle tjenester basert på labels, som f.eks. å ta ned alle tjenester som er merket med "test". Dette var ikke noe gruppen fikk stor bruk for i testmiljøet, mest på bakgrunn av hvor få applikasjoner som ble administrert. Likevel kan gruppen være enige i dette punktet pga. hvor ineffektivt det er å behandle containere bare etter navnet på pods. I et arbeid på større skala ville labels blitt brukt mye mer liberalt.

- Definer "imagePullPolicy"

  Kubernetes sin default-verdi for image-henting er "IfNotPresent". Denne sørger for at Kubernetes bare henter container-bildet om det ikke allerede finnes. I diverse testmiljø og utrullinger vil det ofte være nyttig å kunne bruke en ny policy som "Always" eller med selvdefinerte tags som ":latest". Tags som ":latest" vil ikke anbefales for utrulling i produksjon for det kan gjøre det vanskelig å holde styr på hvilket container-image som faktisk er i bruk. Igjen på grunn av størrelsen på testmiljøet og hvordan applikasjonen til gruppen ble rullet ut så var ikke dette et viktig punkt når gruppen drev med arbeidet sitt. Punktet blir likevel tatt med for å understreke at det lett kan påvirke større Kubernetes-oppsett.

4.2.7 Anbefalt videre arbeid

Om en skal ta en lokal Kubernetes-løsning til å være produksjonverdig vil en del arbeid kreves utover det dette prosjektet fikk tiden til å oppnå. Anbefalingene gruppen kommer med er av stor grad basert på Kubernetes sin egen dokumentasjon om "on-prem"-krav for bedriftsløsninger(Wong & Gasch, 2018).

Lastbalansering

I prosjektet har gruppen lastbalansering på to områder. Det første området er last mot pods som kjører på clusteret, dette ordner Kubernetes med Kube-Proxy. Det andre området er hvordan last utenfra skal nå tjenestene inne på nettverket, noe som ble løst med MetalLB. Programvaren gir automatisk tjenester som er av typen "LoadBalancer" eksterne IP-adresser og kan tilby lastbalansering på lokale Kubernetes-installasjoner.

```
kub@Master-Node:~/metallb$ kubectl get svc --all-namespaces
NAMESPACE       NAME                  TYPE           CLUSTER-IP       EXTERNAL-IP
app             win-mikrotjeneste-1   NodePort       10.110.100.41    <none>
app             win-mikrotjeneste-2   NodePort       10.107.159.234   <none>
app             win-webapp            LoadBalancer   10.110.147.228   129.241.99.5
```

Figur 11: Tildeling av ekstern IP

MetalLB ble konfigurert til å bare fungere bare på "layer 2 mode"(MetalLB, u.d.-b). De andre alternativene er mer egnet for større oppsett, dermed ville ikke gruppens smalere testmiljø kunne stått for hensiktsmessig testing av konfigurasjonen. I "layer 2 mode" delegerer MetalLB en node til å motta trafikk for en Kubernetes-service (som brukes til å nå pods på clusteret) og lar Kube-Proxy videre balansere lasten mellom servicen sine pods. MetalLB kan så tilby failover ved å automatisk bytte den delegerte noden om den slutter å svare.

Problemet med denne løsningen er at noder kan bli "bottlenecked". En bedre løsning for større oppsett som vil ta i bruk MetalLB vil være å sette seg inn i deres "BGP mode". En må huske på at MetalLB er et program som fortsatt er i betafasen, så prosjektgruppen vil ikke blindt anbefale dette som en komplett løsning. Programvaren er likevel et spennende prosjekt som kan tilby aktører muligheten til å bruke fysiske rutere for å takle lastbalansering. Å bruke en skyplattform for lastbalansering er ut ifra prosjektgruppens forståelse en mer ferdigstilt løsning for Kubernetes-oppsett. De mest populære alternativene her er Google Kubernetes Engine, Amazon Elastic Container Service for

Kubernetes og Azure Kubernetes Service. Disse løsningene er integrerte med clusterene en har i skyen, så det havner utenfor denne bacheloroppgaven.

Autoskalering

I Kubernetes snakker en gjerne om to forskjellige typer skalering, horisontal og vertikal. Vertikal skalering vil si å tildele mer eller mindre ressurser til en tjeneste etter hva den har behov for, og denne er allerede implementert i Kubernetes som standard.

Horisontal skalering er derimot ikke noe som er konfigurert fra før av og må settes opp for hver enkelt tjeneste som trenger dette. Det er flere metoder en kan følge for å utføre dette, deriblant programvare eller gjennom kommandolinjen.

Eksempelvis kan tidsbasert skalering enkelt implementeres gjennom å sette opp crontab på linux-masteren ved hjelp av kommandoen "crontab -e". En kan eksempelvis legge inn følgende skaleringskommandoer som øker til fire instanser klokka 7 og senker antallet til to klokka 18:

```
* 7 * * * kubectl scale deployment win-webapp --
   replicas 4

* 18 * * * kubectl scale deployment win-webapp --
   replicas 2
```

En mer dynamisk og fleksibel løsning er naturligvis å bruke programvareløsninger som håndterer skalering basert på IT-administrasjonens parameter. Autoskaleringen i Kubernetes krever at en setter opp overvåkning av de tjenestene det gjelder, og oppgir ressursbegrensninger til hver enkelt pod. I gruppens testmiljø har ikke en hatt mulighet til å teste autoskalering med andre verktøy enn Kubernetes sin Horizontal Pod Autoscaler (HPA). Dette på grunn av problemer knyttet til Kubelet på Windows-nodene og hvordan denne gir ut info om containernes ressursbruk. Siden gruppens utførte testing skjedde på et såpass lite oppsett i forhold til større IT-driftmiljø, så kan ikke gruppen bedømme om verktøyet egnes for større leveringstjenester av IT-systemer.

Prosjektgruppen ønsket i utgangspunktet å bruke Kapacitor i TICK-stacken

til å utføre autoskalering basert på egendefinerte TICKscript. Dette var ønsket på bakgrunn av at TICK allerede benyttes for overvåking, så en kunne spart seg for å ha en metrics-server på clusteret om Telegraf, i kombinasjon med InfluxDB og Kapacitor, kunne uføre den samme jobben.

Problemene knyttet til dette var i stor grad at forespørsler brukte for lang tid på å få respons, eller at dataene som ble returnert ikke inneholdt info som CPU-, minne- og nettverksbruk. Dette resulterte i at TICK ikke mottok data, eller at skaleringsgrunnlaget ble gjort på uønsket basis. Spesielt ettersom CPU-, minne- og nettverksbruk er de mest logiske enhetene å skalere på. Av denne grunn så ikke gruppen på det som veldig praktisk å implementere denne funksjonaliteten ettersom den ikke kunne testes som forventet.

Om en likevel har ønske om å inkludere dette i clusteret sitt vil det anbefales at en er på utkikk både på GitHub, i Kubernetes sin dokumentasjon og InfluxData sine publiseringer. TICK-stacken er mer finpusset på Linux, der en bl.a. kan kjøre Telegraf som en "side-car"(Arbezzano, 2018). På denne måten kjøres Telegraf i samme pod som applikasjonen en skal overvåke, så en får rask og direkte ressursbruk som sendes videre. Prosjektgruppen forventer at disse funksjonalitetene vil overføres til Windows etter hvert.

Maskinstruktur

Prosjektets oppsett har basert seg på en Kubernetes-Master på Linux og tre arbeidsnoder som kjører Windows. Dagens Kubernetes-utgivelser krever en rekke Linux-applikasjoner for å gjøre prosjektets løsning kjørbar, derav deler av TICK-stack og hele MetalLB. Prosjektgruppens anbefaling vil være å fordele disse oppgavene utover til Linux-noder på Kubernetes-clusteret.

Et annet alternativ er å separere "etcd" fra master-noden. Etcd er Kubernetes sitt lager for clusterene sin data, noe som helst burde sikkerhetskopieres. For større oppsett anbefaler Kubernetes å sette opp et eget etcd-cluster(Kubernetes, 2018).

For fysiske hostmaskiner har prosjektet gjennomført Kubernetes sin anbefaling om å gjøre maskinen til en hypervisor. Dette gir løsningen mye større fleksibilitet og sparer Kubernetes for å "flagge" ulike fysiske maskiner og i tillegg bringe muligheten for å isolere kritiske komponenter. En videre anbefaling, som ble for krevende for prosjektets testmiljø, var å ha tre redundante rutere i VM-er, samt HA-løsninger for master-noden.

Backup og tilgjengelighet

Kubernetes har kommet med en rekke anbefalinger angående oppsett som ønsker høy tilgjengelighet:

- Etcd/master-noder: 3, 5 eller 7 instanser:

  Etcd er ifølge Kubernetes det første en burde tenke på å sikkerhetskopiere for Kubernetesoppsett i produksjonsmiljø. Det er tross alt bare denne komponenten som står for den vedvarende lagringen av data for Kubernetes-komponenter og tjenester. Det som blir sett på som "best practice" er å ha fem etcd-noder, ettersom en da kan utføre vedlikehold på en og likevel ha failover og en eldre versjon klar. Tre etcd-noder er et minimumskrav for produksjonsmiljø, og syv er bare anbefalt for enorme miljø med flere "availability zones".

  På samme grunnlag som etcd vil en helst ha et oddetall med masternoder der minstekravet for høy tilgjengelighet er 3. Master-noden er host for Kubernetes-clustere og er en kritisk komponent for oppetid. Etcd er som default på master-noden, men dette punktet går ut ifra at en har fulgt "best practice" om at bedrifter har et isolert etcd-cluster.

- API Server: 2+ instanser:

  API-serveren er grunnlaget for kommunikasjon i et Kubernetes-oppsett ettersom alle forespørsler gjennomføres med HTTP. Uten denne komponenten kan ikke pods administreres.

- Kube-scheduler: 2+ instanser:

  Denne komponenten plasserer pods ut på nodene i clusteret, uten denne vil ikke utrulling eller pod-prioritering kunne utføres.

- Kube-controller-manager: 2+ instanser:

  Uten denne vil ikke kontroll-løkker holde styr på om systemets verktøy er i orden.

- Cloud-controller-manager (CCM): 1 instans:

  Ved bruk av en skyleverandør vil integrasjonen av dens tjeneste bli ødelagt om kontrolleren går ned.

- Add-ons: varierer:

  En siste anbefaling Kubernetes kommer med er å evaluere hvilke tilleggsverktøy en har lagt til i oppsettet sitt og kartlegge dets behov for sikkerhetskopiering.

Sikkerhet

- Certificate Authority (CA)

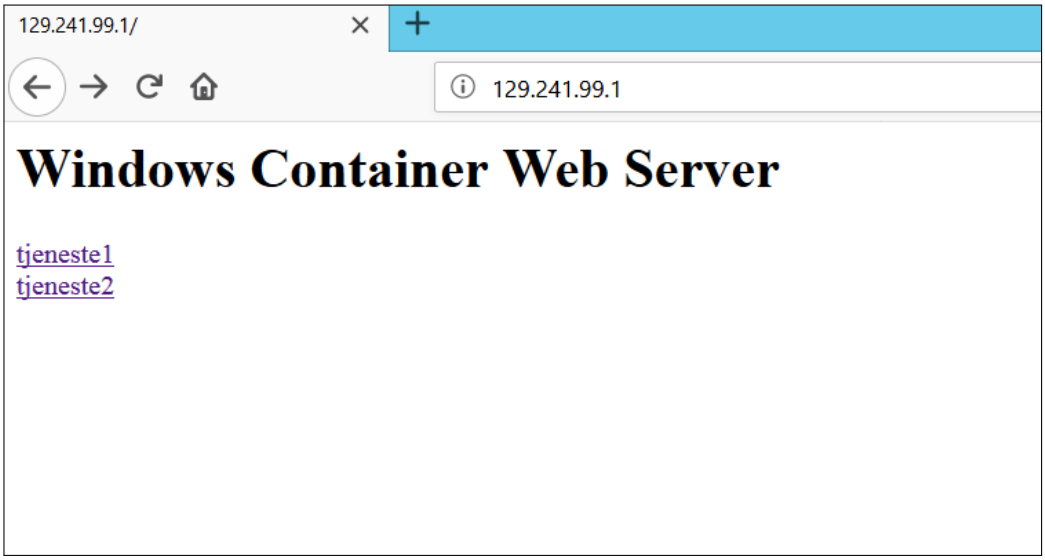
  Kubernetes har et innebygd oppsett for CA i hvert cluster en oppretter. Dette må konfigureres med Controll Manager, API-server, scheduler, kubelet client, kube-proxy og egne administrator-sertifikat. Dette vil begrense hvem som har tilgang til clusteret og hva en har rettigheter til å utføre, noe som er essensielt for Kubernetes siden det er API-drevet. Om en uvedkommen person får tilgang på en tjeneste vil han i teorien kunne sende API-kall videre inn i clusteret.

- Containerscanning

  En vil i utgangspunktet aldri kjørt ut containere i produksjon som hentes direkte fra internettet, dermed vil en ofte ende opp med et lokalt "image"-lager. Med et slikt oppsett anbefales det at det implementeres muligheter for image-scanning, sikkerhetsscanning, aksesskontroll og tester for utrulling og tilbaketrekking av containere. I tillegg kommer behovet for rutiner for oppdatering av programvare, virtualiseringsplattform, operativsystemet og Kubernetes.

4.3 Prosjektets webapplikasjon

Prosjektets webapplikasjon er enkelt bygd opp av tre forskjellige PowerShell-script som kjører som hver sin egen tjeneste. Den ene tjenesten, kalt win-webapp, kjører en ganske så enkel webtjener som er satt opp med en ekstern IP (utenfor Kubernetes-clusteret). Innholdet på websiden denne viser til to andre sider på webtjeneren som heter tjeneste 1 og tjeneste 2. Innholdet på disse sidene vil være hentet fra de to andre PowerShell-scriptene.



Figur 12: Hovedside i webapplikasjonen

De to andre tjenestene kjører hver sin mikrotjeneste, og leverer info til webappen når denne etterspør det. Den første av de to vil returnere ei liste som viser hvor mange ganger den har blitt kontaktet, og på hvilket nettverkskort:

Figur 13: Tjeneste-1 i webappen

Den andre av de to tjeneste vil returnere en kort tekst som vil inneholde pod-ens navn:

Figur 14: Tjeneste-2 i webappen

Kontakten mellom de forskjellige pod-ene vil foregå ved at webapp-en vil kontakte clusteret sin DNS og etterspørre IP til tjenesten som i gruppens tilfelle heter "win-mikrotjeneste-1.app.svc.cluster.local" og få returnert adressen til tjenesten og ikke en individuell pod. Når en så kontakter denne IP-adressen vil Kubernetes rute forespørselen til en passende container av de som er tilgjengelige. Dette gjør at en enkelt kan skalere enkelte av tjenestene uten at alle må oppskaleres å bruke unødig mye av systemressursene. Dette betyr at om en f.eks. har stor pågang på tjeneste 1 så vil Kubernetes kunne oppskalere den individuelt.

4.4 Overvåkning med TICK

I et driftsmiljø vil overvåking av tjenester være kritisk for å forsikre oppti-den på tjenestene en leverer. Med hjelp av TICK kan IT-avdelingen effektivt identifisere problem i container-tjenester. TICK består av de nevnte 4 komponentene:

- **Telegraf** - Henter data fra noder og prosessene på disse
- **InfluxDB** - Database for informasjonen som hentes
- **Chronograf** - GUI for overvåking
- **Kapacitor** - Prosesserer data og sender ut varsler basert på fastsatte regler

4.4.1 Telegraf

Telegraf kjøres som en "service" på Windows-nodene og i container på Linux master-noden. Som tidligere nevnt støtter Telegraf å kjøres som en "sidecar"(Arbezzano, 2018), altså i pods sammen med containerne den skal overvåke. Dette var ikke kompatibelt med Windows-containere ved utføringen av prosjektet. Da Kubernetes v.1.14 kom ut underveis i prosjektet viste det seg å fungere noe bedre med overførselen av metrikk fra Windows-noder. Med Linux-containere var oppsettet feilfritt fra starten av, så det vil naturligvis være noe å ta til betraktning om en velger å sette opp et Kubernetes-miljø med TICK. Influxdata (utviklerne av TICK) har vist et klart mål mot å videreutvikle plattformen sin til å være godt kompatibel med Kubernetes, så bedre kompatibilitet med Windows er noe en kan forvente.

Telegraf konfigureres i sin egen ".conf"-fil, her vil en bl.a. ha muligheten til å presisere hvilke typer data en vil ha fra Docker, hvilke data en vil ha fra Kubelet (Kubernetes sin agent på noden) og til slutt hvilken InfluxDB-database en skal sende informasjonen til. Konfigurasjonsfilen fra prosjektet så eksempelvis slik ut:

Conf:

```
[agent]
        hostname = "Win-Node-01"
        flush_interval = "15s"
        interval = "15s"
[[inputs.cpu]]
[[inputs.mem]]
[[inputs.system]]
[[inputs.disk]]
[[inputs.diskio]]
[[inputs.net]]
        fieldpass = [ "bytes_*"]
[[inputs.win_services]]
[[inputs.docker]]
        endpoint= "ENV"
        gather_services = false
        timeout = "15s"
        perdevice = true
        total = false
[[inputs.kubernetes]]
        url = "http://127.0.0.1:10255"
        response_timeout = "30s"
[[outputs.influxdb]]
        database = "telegraf"
        urls = ["http://192.168.137.50:31460"]
        username = "telegraf"
        password = "Password2"
```

De øverste input-ene er for systeminfo, noe en kan hente og sende til InfluxDB på samme måte som de andre elementene. En faktor en må notere seg om en skal gjennomføre en slik løsning i et reelt miljø ligger under "inputs.kubernetes". URL-en som sjekkes er nodens utrygge "read only"-port for Kubernetes, siden porten ikke krever noe verifikasjon. Ideelt ville en lyttet på port 10250, som krever verifikasjon, i tillegg til et riktig oppsett med sertifikat.

4.4.2 InfluxDB

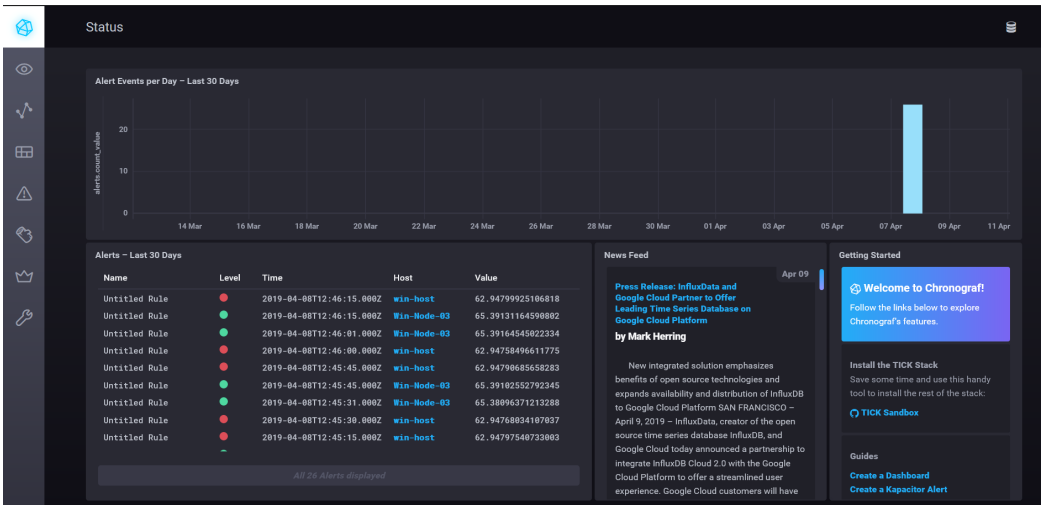
Etter å ha fullført installasjonen av TICK (enten ved hjelp av gruppens installasjonsscript eller manuelt) så vil ikke InfluxDB kreve noe spesifikk konfigurasjon for å fungere med Kubernetes-inputs. Den eneste anbefalingen gruppen kan komme med er at en gjerne vil ha flere instanser av databasen i et større driftsmiljø. Dette nevnes både på bakgrunn av sikkerhetskopiering og i tilfeller hvor en av InfluxDB-maskinene feiler.

4.4.3 Chronograf

Det er i Chronograf en presenterer Telegraf sine videresendte data fra nodene en vil overvåke. Programvaren er delt inn i 8 ulike menyer som en navigerer mellom i menyen til venstre.

Status

Status er det første som møter brukeren når en kobler seg til tjenesten i nettleseren. Siden gir en oversikt over hvilke varsler som har blitt satt i gang av Kapacitor over en gitt periode.

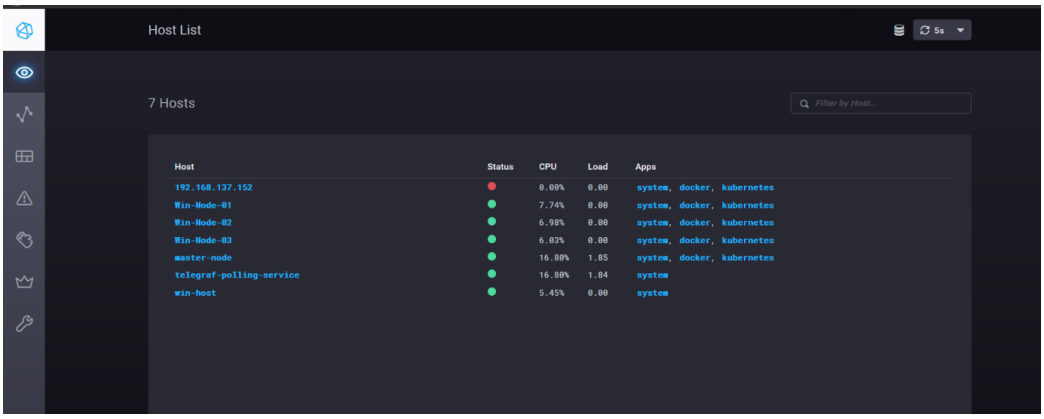


Figur 15: Status-side i Chronograf

Host List

Dette vinduet gir brukeren en oversikt over hvilke maskiner som overvåkes med Telegraf, samt hvilke plugins som er inkludert. På nodene i Kubernetes-

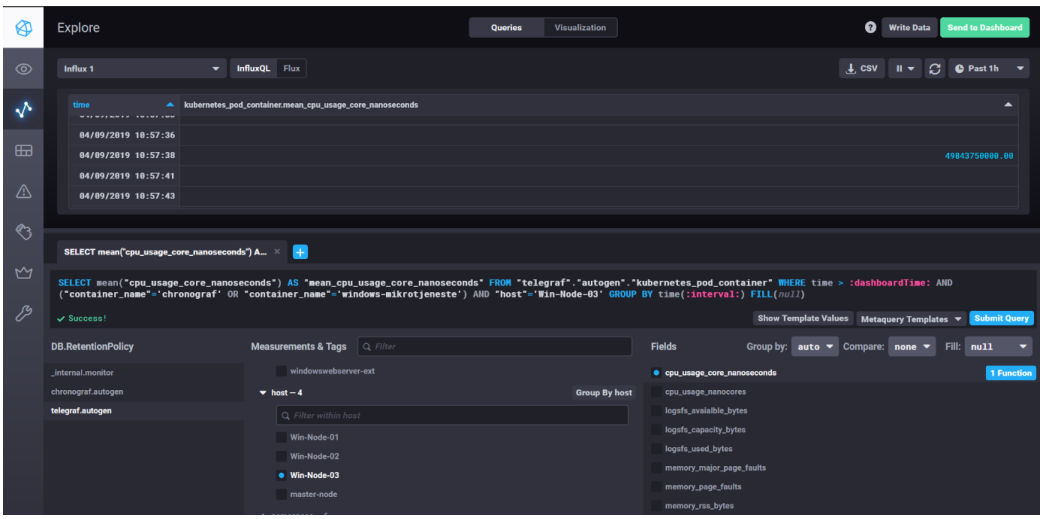
clusteret har prosjektgruppen inkludert Docker og Kubernetes plugins for å vise oversikt over containere og pods.



Figur 16: Host List-side i Chronograf

Explore

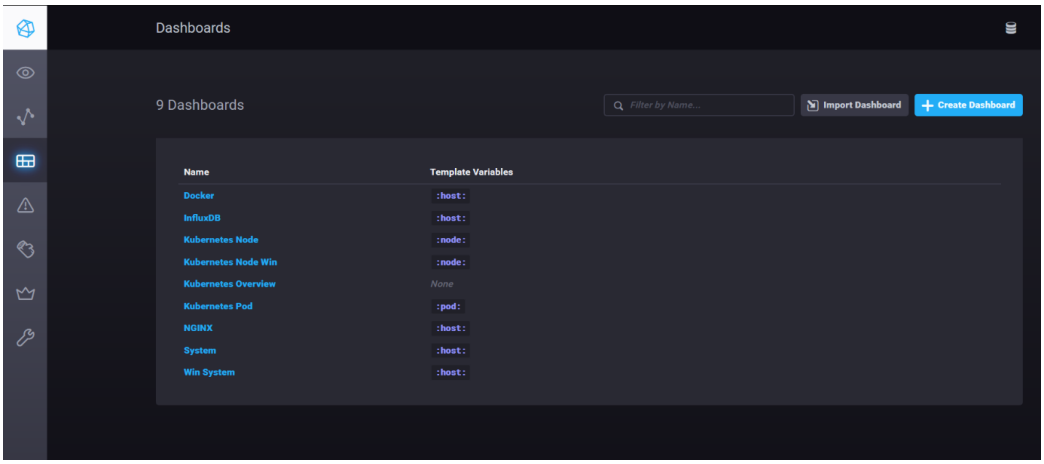
I dette vinduet har brukeren mulighet til å sende spørringer til oppgitt database. For å gjøre det enklere for brukeren kan ønskede spørringer velges med brukergrensesnittet, som demonstreres i skjermdumpen under hvor pods sin CPU-bruk på Win-Node-03 hentes fra Telegraf sin database i InfluxDB.



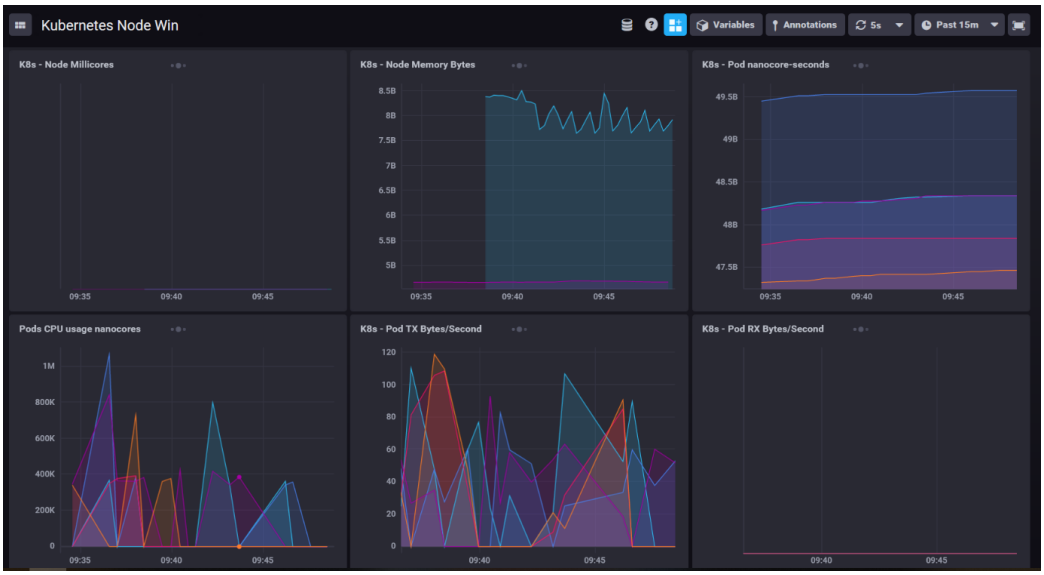
Figur 17: Explore-side i Chronograf

Dashboards

For å gjøre overvåkningen effektiv så har InfluxData laget muligheten for å lage egendefinerte dashboards hvor en kan velge hvilken data som skal være synlig. Eksempelvis lagde prosjektgruppen et dashboard for Windows-noder på clusteret. På skjermdumpen under vises nodens minnebruk, pods sin CPU-tid og ressursbruk og hvor mye datatrafikk pods sender ut.



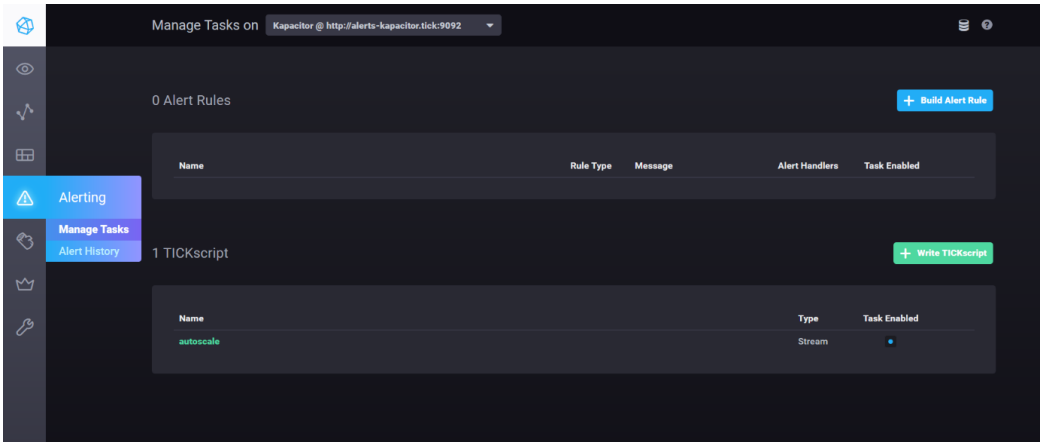
Figur 18: Dashboards-side i Chronograf



Figur 19: Ressursbruk på Windows-noder

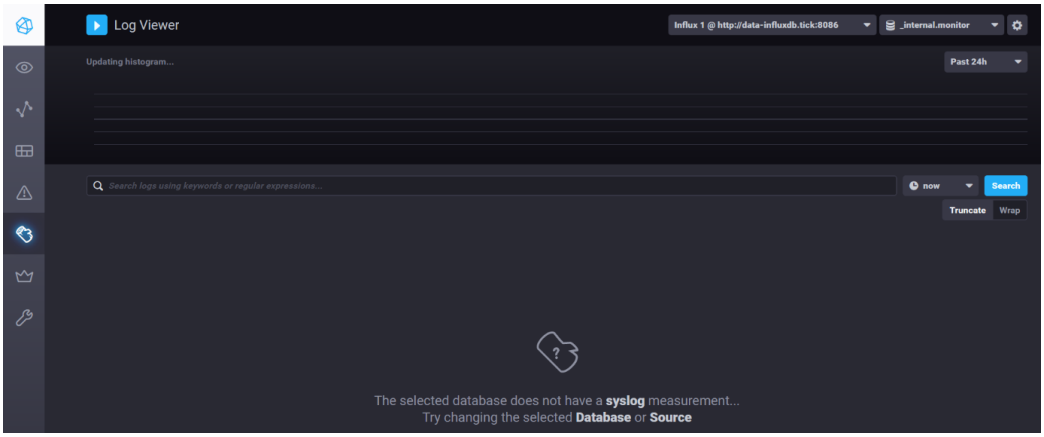
Alerting

Alerting er grensesnittet mot Kapacitor. Her kan brukeren opprette alarmer basert på f.eks. ressursbruk og hvordan brukeren skal varsles. Kapacitor kan i tillegg konfigureres til å utføre handlinger automatisk med hjelp av InfluxData sitt scriptspråk "TICKscript"(Bang & Anderson, u.d.).



Figur 20: Alerting-side i Chronograf

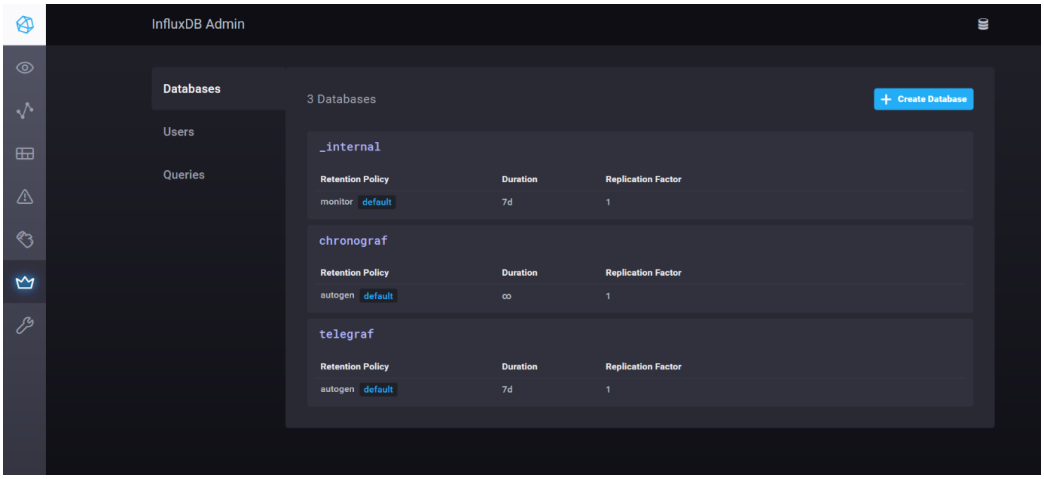
Log Viewer Her kan IT-avdelingen hente logger fra InfluxDB for å vurdere eventuelle feil som oppstår i oppsettet.



Figur 21: Log Viewer-side i Chronograf

InfluxDB Admin

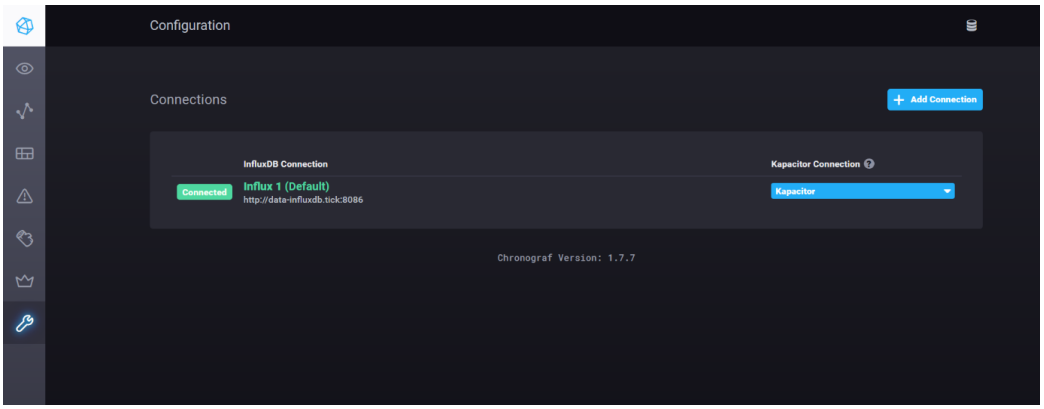
En side for administratoren av databasen hvor nye databaser kan opprettes og nye brukere og rettigheter kan legges til. I tillegg kan administratoren se hvilke nåværende spørringer som gjøres mot databasene, der en vil ha muligheten til å avslutte ineffektive spørringer som bruker mye ressurser.



Figur 22: InfluxDB Admin-side i Chronograf

Configuration

Til slutt i Chronograf finner man konfigurasjonssiden. Her vil en kunne se hvilke tilkoblinger TICK-stacken har for øyeblikket, samt muligheten til å koble til flere eksisterende tjenester (f.eks. andre InfluxDB-instanser).

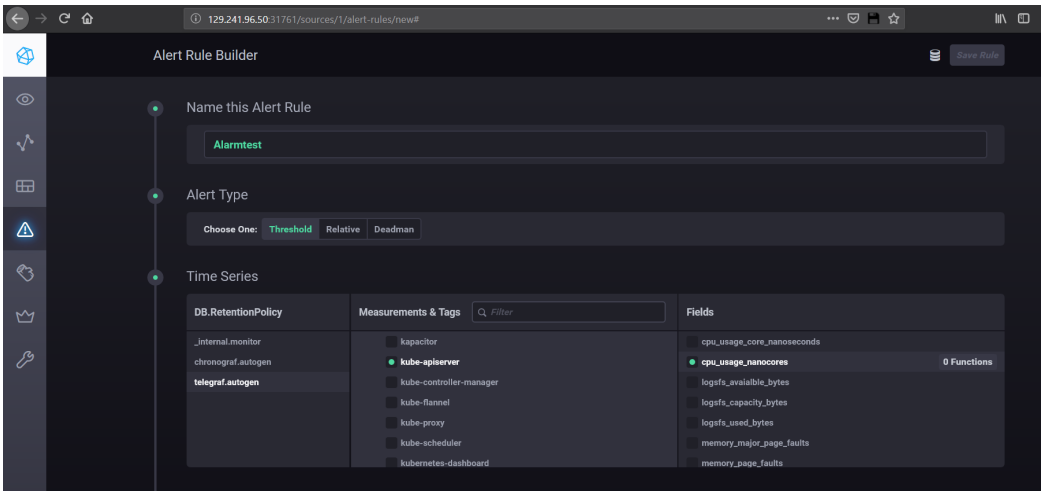


Figur 23: Configuration-side i Chronograf

4.4.4 Kapacitor

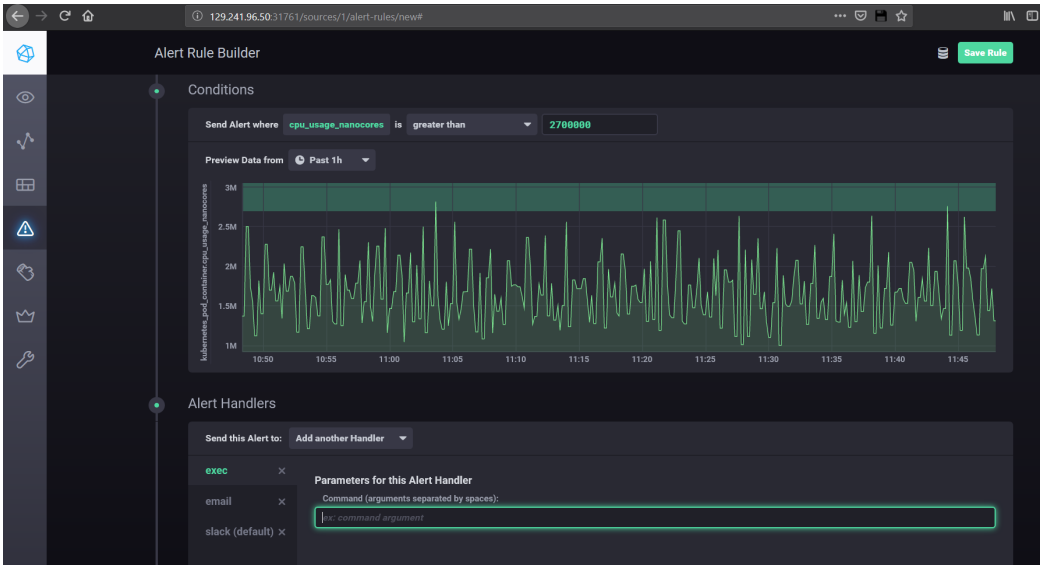
Kapacitor er som nevnt verktøyet en bruker for å sette opp varsellamper og hvordan systemet skal reagere om en lampe går. Typisk vil Kapacitor informere de ansatte på et kommunikasjonsverktøy, f.eks. Slack eller epost. Alternativt vil Kapacitor kunne sende beskjeder over TCP og HTTP eller direkte til loggfiler. Verktøyet kan i tillegg reagere med å kjøre kommandoer, så en kan eksempelvis sette i gang et Bash-script som videre utfører oppgaver på maskinen.

En oppretter nye alarmer i "Alerting"-vinduet. Her vil en kunne gi alarmen et navn, velge alarmtypen og hvilken database og metrikk alarmen skal baseres på. Alarmtypene består av "Threshold" som setter statiske maks- og minsteverdier, "Relative" som bruker prosentbaserte verdier og "Deadman" som kan brukes som en dødmannsknapp for tjenester.



Figur 24: Side for å opprette Alerts i Chronograf

Videre kan en bruke den valgte metrikken til å velge parameter for at Kapacitor skal utføre en handling. Her blir dette demonstrert med "kube-apiserver", som er en av Linux-tjenestene som kjøres på clusteret.



Figur 25: Varsels-alternativer i Chronograf

På grunn av problemet mellom Windows-noder og Kubelet som gruppen diskuterte i punkt 4.2.7, "Anbefalt videre arbeid", så kunne ikke gruppen bruke metrikk fra Windows-pods. Kapacitor skal i teorien kunne brukes for å automatisk skalere tjenester med hjelp av "exec"-alarmer som vist i figuren over. Dette ble dessverre ikke aktuelt siden oppgaven ikke baseres på Linux-noder, hvor Kubelet-problemene er langt mindre fremtredende. Autoskalering med Kapacitor ble dermed ikke mulig å gjennomføre for prosjektgruppen grunnet manglende metrikk.

For å videreutvikle Kapacitor kan verktøyet konfigureres til å bruke egendefinerte "TICKscript", som er InfluxData sitt eget scriptspråk for å bestemme handlinger basert på data. Det var disse scriptene som skulle være utgangspunktet for hvordan Kapacitor skulle samhandle med Kubernetes for å behandle applikasjonene på clusteret.

4.5 Sikkerhet

Selv om sikring av containere ikke er hovedmålet med prosjektet så har gruppen likevel tatt utgangspunkt i både NIST og Kubernetes sin dokumentasjon om hvordan en sikrer et containeroppsett. I dette punktet vil prosjektgruppen presentere hvordan de har forholdt seg til anbefalingene og hva gruppen vil anbefale for en videreutvikling av prosjektets testmiljø. I punktet for videre arbeid diskuteres det smale sikkerhetsbildet i oppgaven, samt Kubernetes sin sikkerhetsmessige umodenhet.

4.5.1 NIST

NIST Special Publication 800-190 har som nevnt vært en del av grunnlaget når det gjelder sikkerhetsaspektet i oppsettet. Under vil gruppen diskutere hvordan de har forholdt seg til noen av publikasjonens hovedpunkter(Souppaya, Morello & Scarfone, 2017):

- Bruk containerspesifikke OS-utgivelser for å minske angrepsoverflaten

  Ved å bruke de nyeste versjonene av Docker og Kubernetes har prosjektgruppen bare forholdt seg til moderne container-images både på Windows (Server Core og Nano Server) og Linux (stort sett varianter av Alpine).

- Gruppere containere med like formål, sensitivitet og risiko på samme noder for å få større dybde i sikkerheten

  Dette er et godt sikkerhetsaspekt som ikke ble gjennomført i prosjektet. En kan tvinge pods til spesifikke "kandidat-noder" med nodeSelector og nodeAffinity, men nodeAffinity er per i dag fortsatt i betafasen. Et annet alternativ er å sette opp separate cluster for tjenester som skal eksponeres i større grad. NIST legger vekt på at selv om containerplattformer er flinke til å isolere containere fra hverandre, så vil det være unødvendig å risikere at en eksponert container kan gå ut over mer sensitive containere på noden. Dette ettersom et suksessfullt angrep på en container kan påvirke OS-et til noden, noe alle andre containerne på noden også nytter.

- Ta i bruk container-spesifikke sikkerhetsverktøy og prosesser for images

  - Tradisjonelle verktøy kan ta antagelser om hosten som ikke stemmer med containermodellen. For eksempel kan programvaren i utgangspunktet gå ut ifra at en server kjører et sett med applikasjoner over lengre tid, men flere applikasjoner kan kjøre på forskjellige servere basert på tilgjengelige ressurser.
  - Kan være trøbbel med å identifisere sårbarheter som kan føre til en falsk følelse av sikkerhet.
  - Organisasjoner burde ta i bruk verktøy for å validere sikre konfigurasjoner basert på best practices for sine images. Dette inkluderer en sentralisert rapportering- og overvåkningsløsning som stopper ikke-kompatible images fra å kjøre.

  Sikkerhetsverktøy var noe prosjektgruppen vurderte å se nærmere på

når det praktiske arbeidet påbegynt. Etter at gruppen forhørte seg om prøvelisenser hos enkelte programvareleverandører så konkluderte gruppen med at den tiden som måtte dedikeres til programvaren ikke var hensiktsmessig for oppgaven. Denne bacheloroppgaven vil dermed ikke kunne gi et grunnlag for å bedømme om dette er et nyttig punkt eller ikke. Scanning av Docker-images er i tillegg en tjeneste som finnes på Docker Enterprise. Dette fikk ikke prosjektgruppen testet ut i Docker Community Edition.

- Vurdere å ta i bruk "Trusted Platform Module" (TPM) for å forsikre seg om at hver host sin firmware, programvare og konfigurasjonsdata er troverdig og validert.
  - Dette kan strekkes videre til OS-kjernen og komponenter for å kryptografisk verifisere oppstartsmekanismer, system-images, container-runtimes og container-images.

  Det nærmeste dette prosjektet kom kryptert autentisering var Kubernetes sine egendefinerte "tokens" mellom tjenestene sine. I et reelt IT-driftsmiljø vil autentiseringskravet være langt viktigere enn i prosjektets testmiljø. Dette ettersom det er større mengde med horisontal datatrafikk mellom containere i forhold til VM-er pga. API-kommunikasjon. Å løse en slik problemstilling på hardware-nivå ble et for stort omfang for gruppen å vurdere å ta i bruk, spesielt siden hele testmiljøet er på en og samme fysiske server.

4.5.2 Kubernetes sikkerhetsanbefalinger

Sikring av cluster

I Kubernetes sin dokumentasjon på hvordan en sikrer et cluster(Kubernetes, 2018) baseres de fleste av anbefalingene på å minimalisere tilgangen til plattformens komponenter, ofte med hjelp av kryptert autentisering. I prosjektet ble det ikke jobbet noe med dette utover det Kubernetes setter opp automatisk. Det Kubernetes setter opp er "service accounts" for hver av sine tjenester. Disse vil være bundet til et namespace og autentiseres med "Secrets" som rulles ut sammen med pods. Når en pod så gjør et API-kall vil den kunne autentiseres med å ha riktig "token" knyttet til seg.

Hadde prosjektgruppen hatt et større tidsrom for bacheloroppgaven ville et oppsett for å autentisere gruppens applikasjoner hatt en større prioritet. I testmiljøet hadde ikke gruppens webapplikasjon noen spesielle avhengigheter innen clusteret, så det var ikke nødvendig å autentisere den for å bare kjøre den på clusteret.

Sikkerhets-policy

"Policies" er noe en gjerne oppretter når en skal rulle ut en applikasjon. Dette defineres oftest i samme YAML-fil som en oppretter en deployment, eksempelvis webapplikasjonen i dette prosjektet. I filen kan en f.eks. legge til en ressurs av typen "PodSecurityPolicy"(Kubernetes, 2019e). I denne konfigurasjonstypen vil en bl.a. kunne definere om pods skal være "privileged" eller ikke, noe som forteller Kubernetes om pods skal ha tilgang på maskinressurser. En type policy prosjektgruppen benyttet var "NetworkPolicy" som vist under:

YAML:

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: egress-webapp
  namespace: app
spec:
  podSelector:
    matchLabels:
      app: win-webapp
  egress:
  - to:
    - namespaceSelector:
        matchLabels:
          name: app
  ingress: - {}
```

Nettverket for applikasjonen ble løst på en forenklet måte der den får lov til å sende data (egress) til alle pods i namespacet "app" (der den selv hører til). Når det gjelder innkommende trafikk (ingress) så godtok den alt. Alternativt kunne en satt opp ingress til å ta imot visse adresser med å bruke "ipBlock:" der en definerer godkjente adresserom. Videre kan en legge "-except" under

ipBlock for å ikke tillate et adresserom gjennom til tjenesten.

4.5.3 Anbefalt Videre Arbeid

Sikkerhetsaspektet med bacheloroppgaven ble nedprioritert på grunn av tidsmessige årsaker. Det var i utgangspunktet et punkt gruppen skulle se på om en hadde litt tid til overs, så det påvirker heldigvis ikke prosjektmålet med å vise bruksområdet for Windows-containere. Lesere av oppgaven vil likevel kunne bruke anbefalingene over som et snevert utgangspunkt for hvor mye arbeid som må legges inn i å sikre et Kuberentes-cluster og hva som burde prioriteres.

En må huske på at Kubernetes per i dag fortsatt er i et relativt ungt stadium, spesielt sikkerhetsmessig. Det vil anbefales å ta stramme vurderinger på hvilke sikkerhetsløsninger en velger, samt å følge med på Kubernetes sin utvikling fremover. Et godt utgangspunkt i dag vil være å baseres på Kubernetes sine egne anbefalinger om hvordan en sikrer et cluster, samt både de tekniske og organisatoriske anbefalingene NIST har kommet med. Et moment for vurdering vil være de etablerte lisensbaserte sikkerhetsløsningene, som f.eks. Twistlock, Aqua eller NeuVector.

Referanser

Arbezzano, G. (2018, 03). *Monitoring kubernetes architecture.* Hentet 24.04.2019 fra https://www.influxdata.com/blog/monitoring-kubernetes-architecture/

Bang, S. & Anderson, S. (u.d.). *Introducing the tickscript language.* Hentet 24.04.2019 fra https://docs.influxdata.com/kapacitor/v1.5/tick/introduction/

CNCF. (u.d.). *Quickstart guide.* Hentet 24.04.2019 fra https://helm.sh/docs/using_helm/

Docker. (u.d.). *Get docker ce for ubuntu.* Hentet 08.04.2019 fra https://docs.docker.com/install/linux/docker-ce/ubuntu/

Kubernetes. (2018, 11). *Securing a cluster.* Hentet 24.04.2019 fra https://kubernetes.io/docs/tasks/administer-cluster/securing-a-cluster/

Kubernetes. (2019a, 03). *Configuration best practices.* Hentet 24.04.2019 fra https://kubernetes.io/docs/concepts/configuration/overview/

Kubernetes. (2019b, 04). *Horizontal pod autoscaler.* Hentet 24.04.2019 fra https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

Kubernetes. (2019c, 04). *Horizontal pod autoscaler walkthrough.* Hentet 24.04.2019 fra https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/

Kubernetes. (2019d, 03). *Overview of kubectl.* Hentet 24.04.2019 fra https://kubernetes.io/docs/reference/kubectl/overview/

Kubernetes. (2019e, 04). *Pod security policies.* Hentet 24.04.2019 fra https://kubernetes.io/docs/concepts/policy/pod-security-policy/

MetalLB. (u.d.-a). *Installation.* Hentet 24.04.2019 fra https://metallb.universe.tf/installation/

MetalLB. (u.d.-b). *Metallb in layer 2 mode.* Hentet 24.04.2019 fra https://kubernetes.io/docs/concepts/configuration/overview/

Schott, D., Lang, P., Kudrayvtsev, G. & Tylenda, P. (2018, 09).

Creating a kubernetes master. Hentet 08.04.2019 fra https://docs.microsoft.com/en-us/virtualization/windowscontainers/kubernetes/creating-a-linux-master

Schott, D., Scherer, S. & Pengweiqhca. (2018, 02). *Joining windows server nodes to a cluster.* Hentet 24.04.2019 fra https://docs.microsoft.com/en-us/virtualization/windowscontainers/kubernetes/joining-windows-workers?tabs=ManagementIP

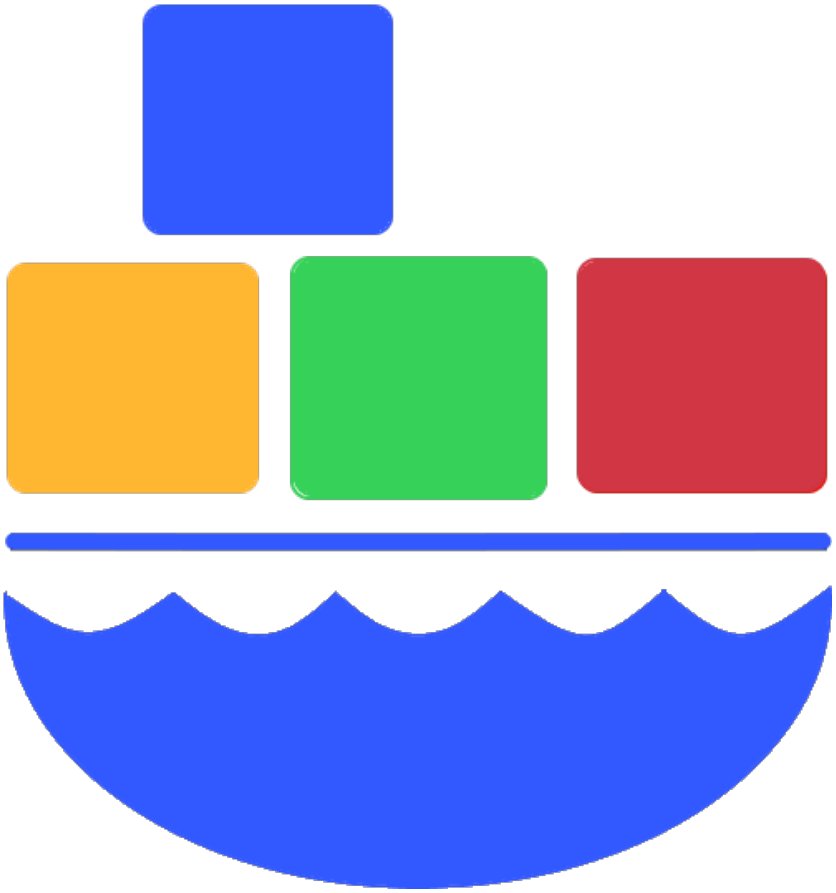
Souppaya, M., Morello, J. & Scarfone, K. (2017, 09). *Nist special publication 800-190 application container security guide.* Hentet 24.04.2019 fra https://www.influxdata.com/blog/monitoring-kubernetes-architecture/

statemigration. (u.d.). *Validation guide - rs5 - kubernetes on windows using flannel.* Hentet 08.04.2019 fra https://statemigration.com/validation-guide-rs5-kubernetes-on-windows-using-flannel/

Wong, S. & Gasch, M. (2018, 08). *Out of the clouds onto the ground: How to make kubernetes production grade anywhere.* Hentet 08.04.2019 fra https://kubernetes.io/blog/2018/08/03/out-of-the-clouds-onto-the-ground-how-to-make-kubernetes-production-grade-anywhere/

Zampolin, J. (2016, 11). *How to spin up the tick stack in a kubernetes instance.* Hentet 24.04.2019 fra https://www.influxdata.com/blog/how-to-spin-up-the-tick-stack-in-a-kubernetes-instance/

Containeroppsett i Windows Server 2019



Sluttrapport

Anders Kvanvig
andekva@stud.ntnu.no

Adis Pinjic
adisp@stud.ntnu.no

Forord

Dette forskningsprosjektet er en bacheloroppgave som markerer slutten på et treårig studieløp i "Informatikk, drift av datasystemer" ved NTNU i Trondheim. Studieprogrammet ligger under "Institutt for datateknologi og informatikk" og tilbyr spesialisering innen IT-infrastruktur. Studentene får et større innblikk i datanettverk og forretningskritiske datasystemer der det er lagt fokus på:

- Hvordan datasystemer brukes i IT-næringslivet
- Hvordan tjenestene kan konfigureres til sine bruksformål
- Hvordan IT-infrastruktur kan sikres fra omverdenen

Under studiet håndteres både Microsoft sine programvareløsninger og domenesystemer, i tillegg til større nettverks- og domeneoppsett i Linux. Dermed vil en etter studiet både ha kjennskap innen Microsoft- og Linuxmiljø, selv om system i Windows har større vektlegging.

I bachelorprosjektet har gruppen i samarbeid med oppdragsgiver Basefarm AS valgt å se nærmere på Windows-containere i Docker, samt administrering av containerne i Kubernetes. Containere bygger videre på konseptet med virtualisering, der containere virtualiserer operativsystemet til en maskin i stedet for maskinvaren dens. På denne måten kan en kjøre applikasjoner og tjenester på fysiske og virtuelle maskiner uten å måtte ha et underliggende OS per applikasjon. Teknologien kan være med på å sette opp mer ressurseffektive oppsett både for fysiske serverparker og på skyplattformer.

Oppgaven har gitt prosjektgruppen en mulighet til å bearbeide ens kompetanse med å ta i bruk ny teknologi, samt hvordan en kan reflektere rundt hvor nyttig løsningen vil være innen moderne IT-infrastruktur. Videre har naturen av et større prosjektarbeid gitt gruppen uvurderlig erfaring innen planlegging og distribuering av tid. På bakgrunn av et godt forarbeid og realistisk målsetting har gruppen klart å oppnå hovedpunktene som var ønskelig å sette opp på systemet. Det eneste gruppen ikke fikk god tid til var sikkerhetsaspektet med Kubernetes, noe som har blitt noe redusert i forhold til hva som var tiltenkt. Dette medførte at drifts-dokumentasjonen rundt det hadde et større fokus på videre anbefalinger i forhold til gruppens sikkerhetskonfigurasjoner. Punktet var ikke et kritisk suksesskriterie for oppgaven, så casebesvarelsen ble ikke redusert i stor grad.

Avslutningsvis vil prosjektgruppen herved takke veilederne for prosjektet for all hjelpen de kunne stille med underveis, noe som har formet prosjektet i riktig retning. Gruppen vil også takke oppgavestiller Basefarm AS som stilte med både veiledning og teknisk støtte for prosjektet, samt en fysisk server som gruppen fikk disponere for testmiljøet. Mange av erfaringene fra arbeidet vil være noe begge medlemmene i prosjektgruppen vil kunne ta med seg videre i arbeidslivet.

Innhold

Figurer

1 Oppgavebeskrivelse

Hensikten med prosjektet var å kunne kartlegge hvilke praktiske muligheter en har med containerteknologi i Windows Server 2019. Prosjektet ble valgt på bakgrunn av samarbeidet Docker har med Microsoft om å utvikle containere som kan være like lettvektige som Linux-motstykkene. Dette har resultert i et container-image kalt "Microsoft Nano Server" som bare består av 232 MB.

Ettersom oppdragsgiveren var en større IT-aktør så skulle besvarelsen rettes mot bruksmål på større skala. På bakgrunn av dette ble det videre bestemt at gruppen skulle implementere en container-orkestrerer i form av Kubernetes, samt et overvåkningsverktøy som kunne hente ressursbruken til Kubernetes sine utrullinger. Om gruppen fikk tid til overs var sikkerhet et ønskelig viderearbeid, der gruppen skulle basere arbeidet på publisert sikkerhetsdokumentasjon både fra Kubernetes og det amerikanske instituttet for standarder og teknologi kalt NIST.

2 Hvordan ble oppgaven løst

2.1 Standarder

All programvare som har blitt brukt under prosjektet består av åpen kildekode med unntak av Windows-lisenser som NTNU står ansvarlig for. Dette ble en løsning for oppgaven på bakgrunn av at både Docker og Kubernetes baseres på åpen kildekode, i tillegg til at prosjektet i utgangspunktet ikke hadde et planlagt forbruksbudsjett.

Når det gjelder standarder så er Docker og Kubernetes fortsatt i et såpass ungt stadium at definitive standarder ikke har blitt fastsatt enda. Den nærmeste prosjektgruppen tok i bruk som kunne nærmet seg en standard er NIST sin publikasjon om containersikkerhet fra 2017.

2.2 Litteratur

Prosjektet baseres i stor grad på programvarenes egen dokumentasjon, deriblant Docker, Kubernetes, Microsoft og InfluxData. All informasjon som ble hentet til oppgaven var da naturligvis hentet over internett.

2.3 Maskinvare

Under prosjektet ble en fysisk maskin satt opp til å være en hypervisor for de virtuelle maskinene som skulle brukes som noder (Windows Server 2019) og Kubernetes Master (Ubuntu 16.04). Maskinen var en "HP ProLiant DL980 G7 Server" med følgende spesifikasjoner:

- 4x Intel Xeon E7-4850 @ 2.00GHz med 10 kjerner
- 128GB RAM
- 3x 300GB i RAID 01 med en backup-disk
- 4x 10Gb nettverkskort, hvorav to av disse var i bruk

2.4 Programvare

Programvaren som ble brukt under prosjektet kan deles i to kategorier: programvare for dokumentasjon og programvare i testmiljøet.

2.4.1 Dokumentasjon

SharePoint - Samarbeidsplattform der rapporter og annen dokumentering har blitt lagret og gjort tilgjengelig for prosjektets veiledere.

Microsoft Word - Microsoft sitt tekstbehandlingsprogram. Verktøyet ble brukt for selve skrivingen av rapporter underveis både lokalt og i samarbeid fordi filer blir oppdatert automatisk i SharePoint.

Word Online - En lettvektig utgave av MS Word som kan brukes som et samarbeidsverktøy for tekstbehandling. Denne varianten av Word ble brukt når prosjektdeltakerne skulle dokumentere samtidig.

Microsoft Project - Et prosjektorganiseringsverktøy opprettet av Microsoft. I prosjektet ble programvaren brukt for å opprette Gantt-diagram for milepæler og tidsfrister under prosjektgjennomførelsen.

Microsoft Excel - Et regneark-program utviklet av Microsoft. Dette ble brukt for timeføring av prosjektgruppens arbeid.

LaTeX - Et typesettingssystem som blir brukt for dokumentproduksjon. I prosjektet har gruppen flyttet over tekst og figurer fremstilt i MS Word over i LaTeX i forkant av leveringen av bacheloroppgaven.

2.4.2 Testmiljøet

Windows Server 2019 - Operativsystem brukt på nodene i gruppens oppsett.

Ubuntu 16.04 - Operativsystemet bruk på Master-noden i Kubernetes-clusteret ettersom vertøyet krever en Linux-maskin til å styre clusteret.

Docker CE 18.09 - Container-motor brukt av gruppen. Denne var nødvendig for å kunne kjøre containere i det tatt på bakgrunn av OS-virtualisering, og var tilstede på både Windows-noder og Linux-master.

Kubernetes v1.14.0 - Programvare for å styre utrulling, rettigheter og tilgang til containere. Lot også gruppen administrere containere brukt i oppsettet.

Metrics-Server - Stod for innsamling av data som dannet grunnlaget for autoskalering av tjenester i gruppens oppsett.

Helm v2.13.1 - Et alternativ for utrulling av tjenester testet i løpet av prosjektoppgaven.

Telegraf 1.9.4 - Delt opp i Telegraf-s og Telegraf-ds som består av samme programvare, men med forskjellig konfigurasjon.

InfluxDB 1.7.4 - En tidsbasert database laget for overvåkning. Denne ble brukt til å lagre data samlet inn av telegraf på alle maskiner i gruppens oppsett.

Chronograf 1.7.8 - Web-basert GUI for overvåkning av tjenester og maskiner. Dette ble av prosjektgruppen brukt til å se status for noder og tjenester, samt administrasjon av Kapacitor.

Kapacitor 1.5.2 - Er et verktøy for å analysere data, og utføre handlinger ved forhåndsbestemte hendelser.

2.5 Rapporter

2.5.1 Forstudierapport

Forstudierapporten omhandler det teoretiske forarbeidet prosjektgruppen gjennomførte i forkant av det praktiske arbeidet i bacheloroppgaven. Dette består av:

- Undersøkelser av markedstrender og teknologisk utvikling
- Teori og analyse rundt valgte løsninger
- Forutsetninger for prosjektet
- Interessentanalyse
- Prosjektets målsetning
- Risikoanalyse

Ved gjennomlesing av forstudiet skal leseren kunne danne seg et bilde av hvorfor prosjektet gjennomføres, hva prosjektet skal bestå av, hvilke risikoer som kan ramme arbeidet og hva prosjektgruppen og oppdragsgiveren ønsker å oppnå. Avslutningsvis i rapporten deler prosjektgruppen sin innsikt om hva en vil anbefale å gå videre med praktisk basert på forskningsarbeidet.

2.5.2 Designrapport

Prosjektets designrapport gir en oversikt over det planlagte praktiske oppsettet basert på forstudiearbeidet. Dette presenteres ved å først beskrive hvilke avgrensinger som ble gjort for å holde prosjektet innen prosjektgruppens

kompetansenivå og tidsramme. Før det praktiske oppsettet forklares gås det så gjennom en kort introduksjon til oppdragsgiverens behov, og hvordan dette gjenspeiles i oppsettet.

Etter at forhåndsinformasjonen er avklart for leseren går deretter prosjektgruppen gjennom løsningen sin på to nivå: overordnet og detaljert. Den overordnede løsningsbeskrivelsen gir en kort og konsis forklaring av:

- Programvare og verktøy som installeres
- Hvilke behov oppsettet vil ha fra hver enkelt komponent
- Krav for å holde oppsettet i drift
- Sikkerhetsvurderinger som burde gjennomgås

Videre i den detaljerte forklaringen går prosjektgruppen gjennom:

- Det fysiske maskinoppsettet
- Full liste over tjenester på maskinene som omhandler oppsettet
- Hva verktøy og programvare gjør i praksis og hvilke behov de dekker
- Hvordan sikring av oppsettet gjennomføres basert på sikkerhetsvurderingen

2.5.3 Driftsrapport

Driftsrapporten er et ekstensivt dokument som inndelt i to hovedpunkter kalt "installasjon og oppsett" og "demonstrasjon og videre anbefalinger". I det første punktet viser prosjektgruppen først en detaljert gjennomgang av hvordan hver delkomponent installeres og konfigureres i testmiljøet. Målet her er at leseren skal kunne følge prosessen prosjektgruppen gikk gjennom for å kunne sette opp et tilsvarende oppsett med riktig konfigurasjon.

I punktet for demonstrasjon og videre arbeid tar prosjektgruppen igjen for seg hver delkomponent, men i denne omgang vises hvordan tjenesten fungerer og hvordan den brukes i praksis. Under hver demonstrasjon vil prosjektgruppen komme med anbefalinger for å kunne gjøre tjenesten driftsverdig i et større miljø. Etter demonstrasjonen vil leseren forhåpentligvis sitte igjen med en god innsikt i hva container-teknologien kan tilby en bedrift basert på oppsettet i prosjektets testmiljø.

2.6 Arbeidsfordeling i gruppen

Under prosjektgjennomførelsen ble arbeid uformelt delegert mellom prosjektdeltakerne basert på personlig interesse og eget initiativ. Med hjelp av jevnlige evalueringer av hva som må utføres innen gitte tidsrammer så klarte gruppen til enhver tid å vite hva som må gjennomføres. Dette gjenspeiles i gruppens ukesrapporter, der daglige oppdateringer ble dokumentert for å lettere kunne hoppe tilbake i riktig tankesett. Denne prosessen hjalp deltakerne til å holde motivasjonen oppe og gav et godt grunnlag for videre planlegging under prosjektet.

Selve gjennomførselen av arbeidet forklares per rapport, ettersom de markerer de store milepælene i prosjektet:

2.6.1 Forstudie

Under forstudiet var det store mengder teori på internett som ble gjennomgått for å danne en overordnet forståelse av teknologien hos prosjektdeltakerne. Dette ble gjennomført ved å først finne reelle containerløsninger og se hvilke verktøy som er ledende på markedet. I denne prosessen leste prosjektdeltakerne hver sin utvalgte publikasjon, der jevnlig dialog og informasjonsdeling dannet en felles forståelse for Docker, Kubernetes, Windows-containere og sikkerhet.

2.6.2 Design

I forstudiet ble det bestemt hvilke mål som er satt og hvilken funksjonalitet teknologien har i teorien. Oppgaven til prosjektgruppen her ble å finne verktøy som kan integreres med Windows-plattformen, samt å finne programvare som kan hjelpe Kubernetes med sine formål. Her fordelte deltakerne arbeidet mellom å se på ulike komplette løsninger og enkelttjenester som kan være relevante for testmiljøet. Om en deltaker avdekket en mulig løsning så ble det diskutert i fellesskap før den eventuelt ble dokumentert i designrapporten.

2.6.3 Drift

Det var først i driftsrapporten at det ble nødvendig med større arbeidsfordeling ettersom kompleksiteten i arbeidet økte i forhold til det tidligere teoriarbeidet. Deltakerne var stort sett samlet under installasjon av noder og Kubernetes-master, men endte opp med å installere og teste hver komponent fra designrapporten individuelt. For å forsikre at begge deltakerne

hadde god nok forståelse innen hver tjeneste så ble det naturlig vist fram underveis om løsningen var suksessfull. Eksempelvis her satt en i gruppen opp overvåkningsplattformen samtidig som webapplikasjonen var under utvikling.

3 Gjennomføring av prosjektet

3.1 Fremdrift

Siden gruppen hadde informasjon om minstekravet for arbeidstimer, samt endelig innleveringsdato, så kunne gruppen estimere hvor lang tid hver milepæl i prosjektet burde ta. Tidsplanen ble tidlig opprettet i et Gantt-diagram der gruppen bestemte følgende tidsplan:

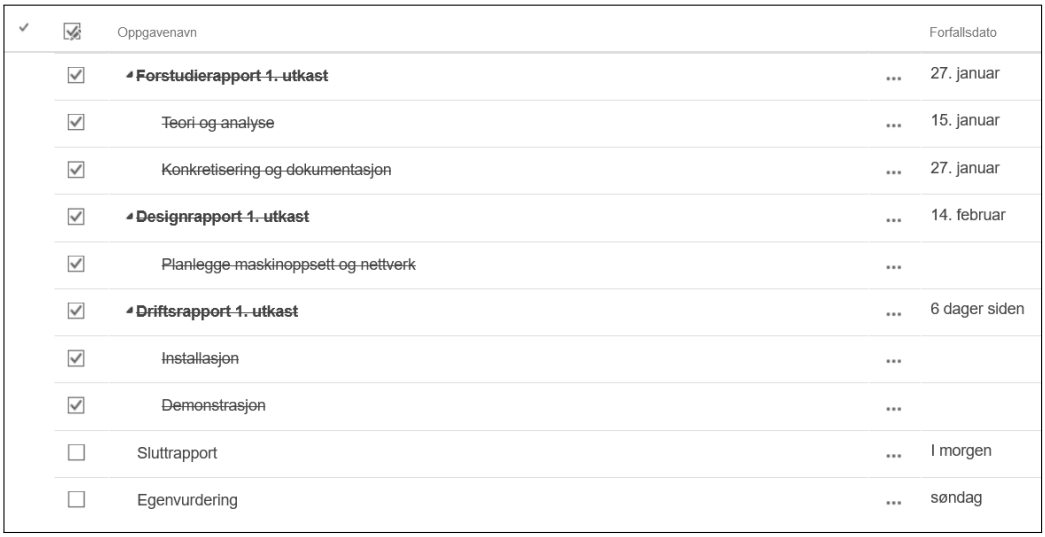
- 16 dager for forstudierapport (07.01.19 - 27.01.19)
- 10 dager for designrapport (28.01.19 - 08.02.19)
- 50 dager for driftsrapport (11.02.19 - 26.04.19)
- 5 dager for sluttrapport (22.04.19 - 26.04.19)
- 2 dager for egenvurdering (01.05.19 - 02.05.19)

Prosjektarbeidet ble utført mandag til fredag stort sett fra 08:00 – 16:00 med unntak av tirsdag. Tirsdager var den ene dagen i uka prosjektgruppen hadde vanlig undervisning, så på denne dagen ble det mer sporadiske arbeidstimer. Lokasjonen for arbeidet var på et reservert bord i 5. etasjen på NTNU Akrinn, mer spesifikt KA-TBL502. Ellers ble det et fåtall av timer gjennomført hjemmefra på eget initiativ.

Prosjektgruppens dynamikk var i god stand under prosjektet, noe som har bakgrunn i at deltakerne har samarbeidet i prosjektarbeid fra og med studiestart i 2016. Dette bidro til at det ikke ble et behov for formalisering av bl.a. arbeidsfordeling ettersom dette vanligvis har skjedd naturlig. Det var av denne grunn at konsekvensen for "splid i gruppe" fikk en relativt høy risikovurdering i forstudiet siden det kunne påvirket arbeidsflyten betraktelig. Gruppedeltakerne fikk i tillegg tidlig fastslått deres ønsker om måloppnåelse i prosjektet, der begge deltakerne ville satse høyt. Dette gav hver deltaker bekreftelse ovenfor hvor mye tid og dedikasjon arbeidspartneren kom til å bringe til prosjektet.

Selve arbeidsflyten i prosjektet baserte seg på å finne en dokumentert løsning som kan passe til oppsettet, forsøke å implementere den, for å så dokumentere løsningen i dens respektive rapport. I praksis resulterte dette i at deltakerne tok vare på lenker til løsningene som ble forsøkt satt opp slik at installasjonsprosessen var lett å replikere i dokumentasjonen. Tidsfristene i Gantt-diagrammet var utgangspunktet for hvilke tidsfrister gruppen skulle

forholde seg til. Diagrammet stemte godt overens med når arbeid ble utført i realiteten, med unntak av driftsrapporten som tok lenger tid grunnet komplikasjoner i oppsettet. I realiteten ble gjennomførselen gjort slik:



Figur 1: Tidsfrister i prosjektoppgaven

3.2 Måloppnåelse

3.2.1 Effektmål

- *Kunnskapen som oppnås om containerløsninger vil kunne bidra til at interessentene for oppgaven kan bedømme Windows-containere til sine bruksmål.*

  Under hele dokumentasjonsprosessen har bachelorgruppen hatt dette effektmålet i baktankene. Før et punkt på en rapport ble skrevet så forsøkte deltakerne å evaluere hvilken kontekst leseren i utgangspunktet vil ha. Dette betydde at gruppen forsøkte å unngå komplisert terminologi tidlig i skrivingen og heller introdusere leseren til begrepene underveis. I forstudierapporten stadfester prosjektgruppen at lesere bør ha forkunnskaper om virtualisering. Dette setter et kunnskapsgrunnlag for at eventuelle lesere skal kunne få mest mulig ut av prosjektets rapporter.

  Prosjektet har underveis blitt evaluert både av veiledere og internt i prosjektgruppen. I forhold til gruppens begrensede kunnskap og erfaring innen større IT-prosjekt så bedømmes det at prosjektet har oppnådd dette effektmålet.

3.2.2 Resultatmål

- *Tjenermaskiner satt opp med Windows Server 2019*

  Prosjektgruppen var vellykket i å implementere Windows Server 2019-maskiner på Kubernetes-clusteret deres. Dagens løsning er derimot ikke ferdigløst, ettersom gruppen fikk trøbbel med Kubernetes sin "node agent" kalt Kubelet når ressursbruk skulle hentes fra containere.

- *Containertjenester i Docker*

  Docker viste seg å være en effektiv "container-engine" og det kom tydelig frem hvorfor programvaren er nærmest en markedsstandard for containere. Uansett om det var en Windows- eller Linux-container så kunne den kjøres på en node så lenge OS-versjonen stemte overens. I gruppens rolle som Kubernetes-administratorer har Docker vært suksessfullt.

- *Administrasjon av containere i Kubernetes*

  Kubernetes har vært et krevende verktøy å bli kjent med og konfigurere fra bunnen av. En typisk Kubernetes-implementasjon i dag vil ofte involvere skytjenester, ettersom containeres skaleringsmuligheter naturlig komplimenterer kapasiteten til skyplattformer. Det har likevel vært en lærerik prosess å sette opp Kubernetes lokalt, noe som kan være aktuelt for et datasenter som vil tilby skytjenester. Implementasjonen i testmiljøet har vært vellykket og det er her de fleste praktiske timene har blitt investert.

- *Fungerende lastbalansering mellom tjenere*

  Lastbalansering ble oppnådd på to nivå. Det første nivået var mellom tjenestene på clusteret, noe Kubernetes gjør automatisk med komponenten "Kube-Proxy" som installeres på hver node. Det andre nivået er for trafikk utenfra som skal nå nodene, der MetalLB delegerte masternoden til å håndtere ruting mellom tjenestene.

- *Automatisk skalering av tjenester*

  Dette ble oppnådd med Kubernetes sin egen "Horizontal Pod Autoscaler" (HPA). En "metrics-server" på clusteret henter ressursbruk fra nodene og skalerer konfigurerte tjenester basert på parametere prosjektgruppen deklarerer. Ønsket her var i utgangspunktet å bruke "Kapacitor" i overvåkningssystemet for å automatisk skalere, noe som blir et punkt for videre arbeid.

- *Demonstrert oppsett for sikring og overvåkning av containere*

  Prosjektgruppen fikk demonstrert hvordan en kan beskytte oppsettet internt med hjelp av policies. På grunn av begrenset tid på grunn av maskinvarekonfigurering og oppsett av Kubernetes så fikk ikke gruppen den tiltenkte tiden til å forske på komplette sikkerhetsløsninger. Altså vil dette resultatmålet være delvis fullført.

- *Dokumentasjon og rammebetingede rapporter*

  Gjennomførselen på dette punktet har stort sett vært godt gjennomført. Prosjektgruppen har tatt dokumentasjonsarbeidet seriøst fra og med første arbeidsdag, noe som har spart gruppen for større etterarbeid mot slutten av prosjektet. Det eneste punktet gruppen bommet på var timelistene, for selv om gruppen noterte arbeidstimene sine daglig så ble ikke arbeidsart definert. Den nærmeste kategoriseringen av arbeidet vil være timefordelingen mellom forstudie, design og driftsarbeidet, der leseren ikke får vite om hvor mange timer som gikk til hvilket underarbeid. Dette er noe gruppen vil ta med seg videre i fremtidige prosjekt.

3.2.3 Prosessmål

- *Få innsikt i bruk og oppsett av containerløsninger i Windows*

  Bacheloroppgaven sitt format som et forskningsarbeid har fungert ypperlig for å bygge opp kompetanse og innsikt innen temaet for prosjektgruppen. Selv om det meste av arbeidstimer gikk i rollen som Kubernetesadministrator, så har prosjektgruppen fått dannet et bilde av hvor bruken av Windows-containere ligger an i 2019. Gruppens evaluering av Windows-plattformen med Kubernetes vil komme nærmere frem i punkt 3.4 "Betraktning i ettertid".

- *Erfaring i hvordan en tar for seg i ny teknologi*
  - Kartlegge nyttige bruksområder
  - Implementasjon av tjenester
  - Formidle kunnskapen videre

  Prosjektgruppen har i bacheloroppgaven gjennomgått det gruppen anser som tre kunnskapsnivå. Det første var da gruppen ikke hadde noe kunnskap om tema og måtte sette seg inn i løsninger basert på teori. Her kikket gruppen på trender i markedet og hvilke verktøy som vil være løsninger for de egenskapene gruppen ville oppnå med oppsettet sitt.

Det andre kunnskapsnivået ble først oppfylt da prosjektgruppen klarte å implementere og konfigurere løsningene til sine bruksmål. Kunnskapen en oppnår her er ofte vanskelig å oversette til personer som er ikke er kjent med teknologien, og det er her det siste kunnskapsnivået blir oppfylt. Under presentasjonsforberedelsen fikk prosjektgruppen sjansen til å ta et steg tilbake og evaluere hva som faktisk ble oppnådd og hvordan en kan presentere dette på en forståelig måte. Alt i alt anser prosjektgruppen at dette prosessmålet har blitt fullført og vil være en uvurderlig erfaring å ta med seg videre i arbeidslivet.

3.3 Timeregnskap

Basert på timelistene så ser timefordelingen per rapport slik ut:

- Forstudierapport
  - Kvanvig: 94,5 timer
  - Pinjic: 98,5 timer
- Designrapport
  - Kvanvig: 81 timer
  - Pinjic: 85,5 timer
- Driftsrapport
  - Kvanvig: 255,5 timer
  - Pinjic: 239 timer

Totalt hadde Pinjic 550.5 timer og Kvanvig 554.5 timer ved avslutting av dokumentasjonsarbeidet. Maksgrensen på arbeidstimer per deltaker skulle i utgangspunktet være 525 timer, men prosjektgruppen fikk godkjent av veiledere å få en utvidet grense. Dette kommer på bakgrunn av maskinvare. Oppsettet prosjektgruppen startet arbeidet på (NTNUs VSphere-miljø) var uforventet tregt, noe som begrenset effektiviteten i starten av det praktiske arbeidet. Da gruppen mottok maskinvare fra Basefarm AS tok det også et par dager å konfigurere denne til å gjeste VM-er med Windows Server 2019. Serveren kunne ikke støtte nyere versjoner av Windows enn Windows Server 2012 R2, der 2019-VMer ble oppnådd gjennom Hyper-V. De ekstra timene som gikk til dette problemet påvirket ikke kvaliteten av det endelige oppsettet, dermed ble ikke vurderingsgrunnlaget endret.

3.4 Betraktning i ettertid

Selv om oppgavenavnet stod som "Containeroppsett i Windows Server 2019", så endte det aspektet med oppgaven å være et delprodukt av å være administrator for et Kubernetes-cluster. Når prosjektgruppen satt i gang med forstudiet så var ikke gruppen klar over at Kubernetes bare kjøres på Linux, og at Windows-maskinene bare vil være "statiske" noder som kjører containere. Dermed ble det et større fokus på de administrative mulighetene i Kubernetes i stedet for utviklingsmiljøet til Docker i denne omgang.

Om en skal implementere et slikt system med erfaringen til prosjektgruppen i baktankene så vil en mer Linux-lenende variant vært den mest stabile. Windows-containere er et pågående prosjekt som kan bli mer relevant om det kommer et behov for å flytte Windows-applikasjoner i Kubernetes-cluster.

Gruppen har ikke hatt noen problemer med selve Docker og Windows-containerne, så en mer stabil løsning i dag vil være å kjøre Docker uten å involvere Kubernetes. En slik løsning vil en kunne gjøre maskinvare mer kostnadseffektiv med å virtualisere OS-et, men en mister naturligvis godene innen administrering med Kubernetes.

Når det gjelder selve prosjektutførelsen så satset nok gruppen for bredt. I utgangspunktet ville gruppen sette av mer tid for sikring av oppsettet, spesielt ettersom casebeskrivelsen peker mot løsninger til en større bedrift. Det viste seg at å konfigurere Kubernetes med Windows-containere var en mer trøblete prosess enn det prosjektgruppen hadde forventet. Dermed ble det ikke mye tid til overs for å besvare punktet for sikkerhet like omfattende. Erfaringen og kunnskapen prosjektdeltakerne sitter igjen med i ettertid er likevel positiv, og vil være noe begge medlemmene i gruppen vil kunne ta med seg videre i karrieren.

4 Videre Arbeid

Stegene som bør tas for å gjøre prosjektets oppsett driftsverdig for en bedrift ble et stort fokus i driftsrapporten. Det vil være flere moment som må løses og hele oppsettet bør gjennomgå ekstensiv testing før en setter tjenester i drift. Når en eventuelt har en komplett Kubernetes-løsning så vil det videre arbeidet bestå av å gjøre utviklingsmiljøet tilgjengelig for resten av bedriften. Både eksisterende og nye applikasjoner må tilpasses Docker, noe som kan optimaliseres ved bruk av DevOps-lag per mikrotjeneste og egne pipelines for utviklingen. CI/CD-implementasjoner som Jenkins som kan være relevante for automatisering av slike løsninger.

De fleste aktører som går over til Kubernetes de siste årene har basert seg på skyplattformer sine Kubernetes-implementasjoner. Om en ikke har et eget datasenter med enorm kapasitet, så vil kanskje ikke Kubernetes sitt potensial for skalering kunne benyttes fullverdig. Så for mange i næringslivet vil det å benytte Azure, Amazon og Google sine Kubernetes-mastere være den mest effektive løsningen.

Et videre punkt for nettverket til Kubernetes-cluster er en ekstern lastbalanserer. Dette tilbys av skytjenester og vil kunne tilby enormt mye større kapasitet enn fysiske løsninger. MetalLB benyttes i gruppens oppsett, men verktøyet er fortsatt i alpha-fasen, noe som ikke nødvendigvis kan anbefales som en løsning for større IT-aktører.

Anders Kvanvig og Adis Pinjic

NTNU
Kunnskap for en bedre verden