NTNU
Norwegian University of
Science and Technology

Containers in Windows Server 2016

Forfattere
Andreas Dukstad
Jan Kristiansen
Åsmund Helland Bu
Svein Edmund Varfjell

Bachelor i informasjonssikkerhet
20 ECTS
Avdeling for informatikk og medieteknikk
Norges teknisk-naturvitenskapelige universitet,

18.05.2016

Veileder Erik Hjelmås

Sammendrag av Bacheloroppgaven

Tittel:	**Containere i Windows server 2016**
Dato:	18.05.2016
Deltakere:	Andreas Dukstad Jan Kristiansen Åsmund Helland Bu Svein Edmund Varfjell
Veiledere:	Erik Hjelmås
Oppdragsgiver:	Ikomm
Kontaktperson:	Dag Olav Nilsen
Nøkkelord:	Docker, Windows Server 2016, Containere, containerteknologi, ytelsestesting
Antall sider:	164
Antall vedlegg:	11
Tilgjengelighet:	Åpen

Sammendrag:

Denne bacheloroppgaven ble utformet for å besvare Ikomms spørsmål angående implementasjon av Windows Container i Windows Server 2016. Gruppen har i denne oppgaven utforsket funksjonaliteten til containere i Windows Server 2016 - technical preview 4 og arkitekturen bak, og deretter målt ressursutnyttelse opp mot eksisterende virtualiseringsteknologier. Gruppen sitter i etterkant igjen med et inntrykk av at Windows Container ikke er en moden teknologi. Mangelen på et grafisk grensesnitt og RDP-støtte innsnevrer bruksområdene til containere betraktelig. Microsoft har gitt utrykk for at dette heller ikke vil bli støttet i endelig utgave av Windows Server 2016. Windows er i stor grad et grafisk operativsystem, og de fleste programmer har grafiske komponenter som de er avhengig av, noe som forsterker denne problematikken. Under vår testing kommer Windows Container ytelsesmessig ut både svakere og mer ustabil satt opp mot virtuelle maskiner kjørt i Hyper-V, men stort sett med små marginer. Ettersom målet med containere er å få en bedre utnyttelse av eksisterende hardware, tyder dette på at Windows Container ikke er godt nok optimalisert. Med denne begrunnelsen sammen med et manglende behov for horisontal skalering fra Ikomms side, anbefales de ikke å implementere Windows Container i sin serverpark.

Summary of Graduate Project

Title:	**Containers in Windows Server 2016**
Date:	18.05.2016
Authors:	Andreas Dukstad Jan Kristiansen Åsmund Helland Bu Svein Edmund Varfjell
Supervisor:	Erik Hjelmås
Employer:	Ikomm
Contact Person:	Dag Olav Nilsen
Keywords:	Docker, Windows server 2016, Containers, containertechnology, bencmarking
Pages:	164
Attachments:	11
Availability:	Open

Abstract:

This bachelor thesis was designed to answer a series of questions from Ikomm regarding the implementation of Windows Container in Windows Server 2016. This thesis explores the functionality of containers in Windows Server 2016 - technical preview 4, the architecture behind, and compared resource utilization benchmarks to existing virtualization technologies. After finishing the thesis the group is left with the impression that Windows Container is not yet a mature technology. The lack of a graphical interface and RDP support narrows the usecases for containers considerably. Microsoft has expressed that this will not be a functionality in the final release of Windows Server 2016 either. Windows is largely a graphical operating system, and most programs have graphical components that they depend on, which reinforces this issue. During our testing, the performance of Windows Container has been shown to be both weaker and more unstable when compared to virtual machines running in Hyper-V, albeit not by much. However, since the objective of containers is to achieve a better utilization of existing hardware, it suggests that Windows Container is not sufficiently optimized. With this rationale along with Ikomm not being limited by horizontal scaling, we advice them not to implement Windows Container in their server park.

FORORD

Gruppen ønsker å takke vår oppdragsgiver Ikomm for at de gav oss en interessant bachelor-oppgave innen IT-drift. Virtualisering innen drift av applikasjoner og tjenester er teknologi vi har hatt stor nytte av å lære gjennom denne bacheloren. Nå som gruppen har gjennomført en oppgave om containerteknologi, setter vi pris på å ha kunnet sette seg inn i det som nå er i vinden innen virtualisering, og som vi gleder oss til å se mer av i fremtiden. Oppgaven har gitt oss god innsikt i hvor langt containerteknologien har kommet, og hvorvidt den er klar til å ta over for virtuelle maskiner. Gruppen håper at vår utredning av containerteknologien Docker kan hjelpe Ikomm til å vurdere om dette er en teknologi som kan implementeres og gi fordeler i deres leveranse av sky-tjenester og andre løsninger.
Gruppen vil også takke vår studieansvarlig og veileder Erik Hjelmås som har støttet og tipset oss underveis gjennom vårt arbeid.

Åsmund H. Bu | Andreas Dukstad | Jan Kristiansen | Svein E. Varfjell

Gjøvik 16.05.2016

Terminologiliste

CPU: Forkortelse for Central Processing Unit - prosessor
Daemon: Et program som kjører som en bakgrunnsprosess
Hyper-V: Microsofts virtualiseringsplatform
Hypervisor: Programvare som tillater en maskin å kjøre mer enn ett operativsystem
I/O: Forkortelse for input/output
IOPS: Input/output operasjoner per sekund.
GUI: Forkortelse for graphical user interface, eller grafisk brukergrensesnitt
Kernel: Kjernen i en maskin, håndterer tråder i prosessor, ressursfordeling, filhåndtering osv.
OS: Forkortelse for operativsystem
Patch: "Lapping" av ferdig programvare, typisk for å fikse en feil eller utvide funksjonalitet
PowerShell: En automatiseringsplatform og et skriptingspråk utviklet for Windows
RDP: Forkortelse for remote desktop protocol, tillater en bruker å koble til en annen maskin med grafisk brukergrensesnitt
SQL: Forkortelse for structured query language, programmeringsspråk for databaser
Switch: Nettverkskomponent som styrer datatrafikk mellom ulike noder i et nettverk
VDI: Virtual Desktop Infrastructure, Virtuelle skrivebord kjørt på en server
VM: Forkortelse for Virtuell maskin
Web server: En teknologi som prosesserer og sender nettsider til klienter
YAML: Forkortelse for "YAML Ain't Markup Language"

INNHOLD

FIGURER

TABELLER

Del I

Introduksjon

KAPITTEL

1

INNLEDNING

1.1 Oppgavebeskrivelsen fra Ikomm

Ikomm er en totalleverandør av IKT-tjenester til kunder i privat og offentlig sektor. De spesialiserer seg på alt fra skytjenester og driftstjenester, til klientservice, rådgivning og prosjektstyring [5].

Oppgaveteksten fra Ikomm med spørsmålene denne oppgaven skal besvare lød slik:

> Ikomm er ute etter å få utredet hvordan containerteknologi som Docker vil påvirke organisasjoner som dem. Ikomm bruker i dag primært VmWare, HyperV og Xenserver som plattform som virtualisering, og bør ta utgangspunkt i følgende problemstillinger:
>
> - Hvordan vil Docker påvirke denne plattformen?
> - Er Docker en komplementær teknologi, eller en konkurrent til nåværende plattform?
> - Finnes det referanseprosjekter på bruk av Docker som er relevante for Ikomm?
> - Hva slags fordeler kan Ikomm få ved å anvende denne type tjenester (gevinstrealisering)?
> - Hvordan påvirker Docker leveransen av terminalservere/VDI?
> - Hvordan påvirker Docker leveransen av andre type applikasjoner som økonomisystemer, arkivsystemer o.l.?
> - Hvordan endringer i kompetansekrav kommer som en konsekvens av Docker teknologien?
> - Hvor ligger utfordringene i bruke av Docker, både av organisatorisk og teknisk art?

1.2 Målgruppe

Målgruppen til denne oppgaven er Ikomm og andre bedrifter som er interessert i virtualiseringsteknologi og ønsker å finne ut om containere i Windows Server 2016 er et verdifullt verktøy for deres situasjon.

1.3 Formål

Oppgaven skal besvare spørsmålene nevnt i oppgavebeskrivelsen, i tillegg til å utrede i hvilke tilfeller det vil være riktig å erstatte virtuelle maskiner med containerteknologi. Om containerteknologi er bedre eller ikke, vil vurderes ut fra om det enten er ressursbesparende eller en raskere/lettere løsning i applikasjoner der dette er viktig.

1.4 Rammer

Prosjektet skal forholde seg til containerteknologien Docker som vil være inkludert i Windows Server 2016. Dette er relevant for vår arbeidsgiver ettersom de er sterkt knyttet til Microsoft og har avtaler med dem som gir dem visse goder, men også stiller krav til bruk av Microsoft-produkter. Operativsystem, maskinvare og programvare vil vi få utdelt av arbeidsgiver der vi vil ha vårt arbeidsmiljø. Her vil vi implementere og teste casene vi får tildelt av arbeidsgiver. Videre har prosjektet en tidsramme hvor rapporten skal være ferdig og levert innen 18. mai 2016.

1.4.1 Avgrensing

Dette prosjektet er avgrenset til containere i Windows Server ettersom dette er operativsystemet Ikomm hovedsaklig benytter seg av og er mest interessert i. Containerfunksjonaliteten er videre kun tilgjengelig på Windows Server 2016 Technical Preview 3

og oppover. Technical Preview 4 er den siste offentlig tilgjengelige versjonen av Windows Server 2016 per dags dato, så det er denne som blir lagt til grunn i denne oppgaven, og det tas forbehold om at resultatene fra denne oppgaven kan være missvisende i forhold til den endelige versjonen av Windows Server 2016.

1.5 Utstyr

Utover eget utstyr som er blitt brukt har vi fått tildelt en server fra Ikomm på Lillehammer med følgende spesifikasjoner (tabell 1).

Server:	Dell PowerEdge T610
Prosessor:	2x Quad-Core Intel Xeon E5506 2100 MHz 4mb Cache
Minne:	24 GB DDR3 789 MHz
Harddisk:	1TB 7.2k

Tabell 1: Serverspesifikasjoner

1.6 Øvrige roller

Vår oppdragsgiver er Ikomm AS på Lillehammer, de er en sterk fagressurs innenfor dette feltet. Vi vil få den programvaren og maskinvaren vi trenger for å fullføre dette prosjektet på en god måte.

Førsteamanuensis Erik Hjelmås ved NTNU Gjøvik er vår veileder. Han vil være til stor nytte for oss teoretisk og teknisk innenfor dette feltet, ettersom han er ansvarlig for kurs som har sterke relasjoner til dette emnet.

KAPITTEL

2

VIRTUALISERING

Virtualisering er en teknologi som ble utviklet på 1960-tallet [6]. Denne teknologien gjorde det mulig å bruke programvare som tillater en fysisk maskin å kjøre flere og gjerne ulike operativsystemer samtidig. Dette gir bedre utnyttelse av ressurser som ellers i mange tilfeller ville stått ubrukt. Man kan kontrollere ressursene og fordele dem etter behov blant de virtuelle maskinene på den fysiske serveren.

Virtualisering kan i stor grad ses på som en viktig del av fremtidens server-teknologi. Automatiske og dynamiske IT-løsninger der løsningen selv sørger for at den har nok tilgjengelige ressurser, er et godt eksempel på bruk av virtualisering. Dersom det er høy aktivitet i forhold til tilgjengelige ressurser kan et slikt system selv skalere opp mengden ressurser tildelt hver virtuelle maskin, eller legge til flere maskiner [7]. En slik løsning ville nærmest vært en umulighet uten introduksjonen av virtualisering.

Virtuelle maskiner er basert på vert/gjest paradigmet. Hver virtuelle maskin(gjest) kjører på en virtuell imitasjon av den fysiske maskinen [1]. Dette tillater gjestens operativsystem å kjøre som normalt uten modifikasjoner på selve koden. Gjesten vet ingenting om hvilket operativsystem verten kjører. da den ikke er klar over at den ikke kjører på fysisk maskinvare [8].

Ved oppstart av en virtuell maskin kreves det at et helt operativsystem startes, dette kan føre til tilnærmet lik oppstartstid som på en fysisk maskin. Denne tiden avhenger av hardware, hvilket operativsystem kjøres og hvor mye last som er på maskinen. Det medfører også at en del ressurser, i form av diskplass, CPU-tid og minne går med til å kjøre disse.

2.1 Fordeler med virtualisering

Det finnes mange fordeler for en bedrift, uavhengig av bedriftens størrelse, med å ta i bruk virtualisering. Noen av disse er:

Energisparende

Å gå fra kun fysiske servere og over til virtuelle maskiner vil redusere antall fysiske maskiner drastisk. Dette betyr lavere strømkostnader, både på grunn av færre servere og på grunn av et minsket behov for kjøling i datasenteret.

Mindre datasenter

Da det er behov for et lavere antall fysiske servere, vil dette også bety at en kan redusere mengden nettverksutstyr, racks og generell plass som er nødvendig for å plassere datasenteret [9].

Hurtig oppsett

I stedet for å bruke tid på å kjøpe inn en ny fysisk maskin, vente på levering, sette opp i rack, koble opp og installere operativsystem for hver eneste maskin, kan det gjennomføres på minutter med en virtuell maskin, ved å klone et image eller en annen virtuell maskin som allerede brukes til samme formål.

Forbedret oppetid

Mange virtualiseringsplatformer tillater å flytte virtuelle maskiner fra en fysisk maskin til en annen i løpet av få sekunder, noe som kan bidra til en høyere oppetid [10]. Dersom det skulle oppstå uforutsette problemer med den opprinnelige fysiske serveren, kan en

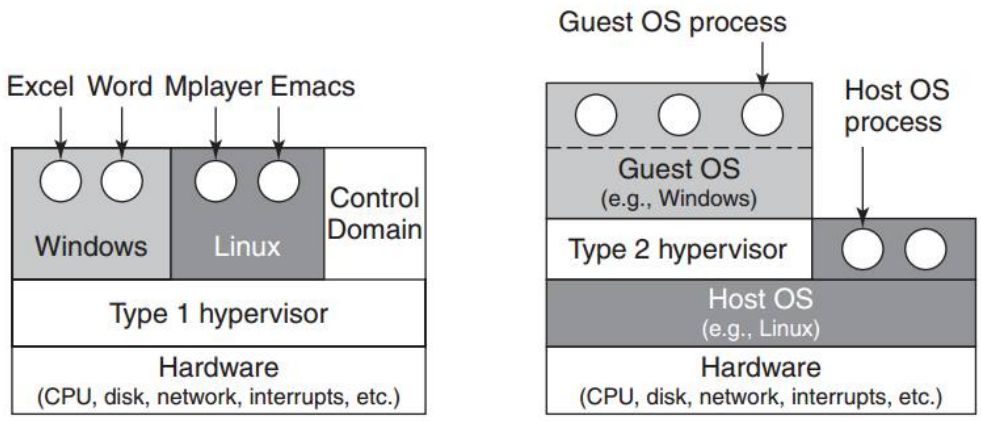
raskt sette opp igjen den virtuelle maskinen et annet sted.

2.2 Hypervisor

En hypervisor, også kjent som Virtual Machine Manager (VMM), er et program som gjør det mulig å opprette flere operativsystemer på samme datamaskin. Operativsystem som kjører i en hypervisor, vil ha en illusjon av at den har datamaskinens prosessor, minne og andre ressurser for seg selv [1]. Det er valgfritt hvor mye ressurser en hypervisor skal få tildelt, men den vil kreve et minimum av ressurser for å kjøre operativsystemet. Det finnes to forskjellige typer hypervisor, kjent som type-1, native eller bare-metal hypervisors, og type-2, hosted hypervisors [11].

På type-1 kjøres hypervisoren direkte på hardwaren [12] (Figur 1). Jobben til denne type hypervisor er å kopiere hardwaren for å kunne opprette virtuelle maskiner. Denne blir kalt bare-metal hypervisor eller native. De første hypervisorene som ble utviklet av IBM på 1960-tallet, var native hypervisorer. Eksempler på hypervisor type-1 er Citrix XenServer, Microsofts Hyper-V, VMware VSphere/ESXi [13].

På type-2 kjøres det et operativsystem som en ordinær datamaskin (Figur 1). Deretter kjøres hypervisoren, som oppretter et nytt operativsystem. Denne hypervisoren tideler de virtuelle maskinene de ressursene som trengs for å kjøre operativsystem, eller det som er nødvendig for å kjøre applikasjonene på de virtuelle maskinene [1]. Eksempler på hypervisor type-2 er VMware Workstation, VMware Player, Virtualbox [14].



Figur 1: Forskjellen mellom hypervisor type 1 og type 2 [1]

Del II

Teori og praksis

KAPITTEL

3

HVA ER CONTAINERTEKNOLOGI?

I tiden før virtualisering ble tatt i bruk for fullt, ble det brukt en fysisk server for hver applikasjon en ønsket å kjøre. Etter at virtualisering ble introdusert og implementert ble det mulig å sette opp flere virtuelle maskiner på hver fysiske maskin for å spare ressurser. I oppstarten av dette var det type 2 hypervisor som var det mest populære valget, altså at en har et operativsystem (OS) kjørende på den fysiske serveren og har hypervisor kjørende på toppen av dette. Med tiden ble det mer aktuelt å eliminere host-OS og kjøre hypervisor direkte på den fysiske serveren, uten et underliggende host-OS (type 1 hypervisor). Dette sparte ressurser og effektiviserte serverdriften enda mer.

En bedrift ville da normalt ha en applikasjon kjørende på hver virtuelle maskin. Hver av disse krever sine egne dedikerte ressurser, for eksempel CPU, minne, I/O og nettverks-konfigurasjon. I tillegg til dette krever hver enkelt VM sitt eget separate operativsystem for å kjøre applikasjonen på.

Det er her containerteknologien kommer inn. I stedet for å ha hver applikasjon kjørende i sin egen virtuelle maskin med sitt eget operativsystem kan en bruke containere [15]. Kort forklart er en container et isolert miljø kjørende på en fysisk maskin eller i en virtuell maskin. Her kan en kjøre en applikasjon uten at det påvirker resten av systemet og uten at systemet påvirker applikasjonen. Applikasjoner som kjører i en container vil tro at den kjører på en egen maskin uten å vite om andre eventuelle containere som kjører på den samme fysiske maskinen [16].

Containerteknologien blir ofte kalt operativsystem-virtualisering. Teknologien gir applikasjonen som kjører på containeren inntrykket av at den har et isolert og uavhengig operativsystem. Containeren tror at alle filene, inkludert OS-filene, kun tilhører seg selv. Minnet inneholder tilsynelatende bare data som er relevante for den kjørende containeren.

Containere som kjører på samme maskin deler det samme operativsystemet, det samme gjelder mange av de kjørende prosessene og filer på maskinen. Det er først når en container gjør endringer som enten medfører at den har en ulik versjon eller ny fil i forhold til de andre containerene at det lages nye unike filer for de områdene som er endret. Resten av operativsystemet og øvrige felles filer forblir delt mellom containerne [17].

3.1 Docker

Docker er programvaren som Microsoft har tatt i bruk i Windows Server 2016 for å kunne bruke containere. Docker ble opprinnelig laget for Linux og er designet for å forenkle prosessen med å lage, distribuere og kjøre applikasjoner. Verktøyet Docker er utviklet for å kunne brukes gjennom hele livssyklusen til en applikasjon, både i utvikling og drift i etterkant [18]. Denne prosessen blir ofte omtalt i forbindelse med utrykket DevOps. For utviklere betyr dette at mer tid kan benyttes til programmering, uten å bekymre seg for hvilket miljø den ferdige applikasjonen skal kjøre på, ettersom man kan sette opp identiske miljøer ved hjelp av en dockerfil [19]. Dette blir skrevet mer i detalj i delkapittel 3.4.5. Når det gjelder driften av applikasjonen i etterkant, vil Docker gi fleksibilitet i tillegg til potensielt å redusere ressursene som kreves til drift ved at containere gir lite overhead.

Containerteknologien kan deles opp i mindre enheter som danner en større helhet, som i Docker:

- Container host: enten en fysisk eller virtuell maskin som er konfigurert med Windows Container featuren. Denne kjører en eller flere Windows Container.
- Sandbox: Når en container startes, blir alle handlinger, systemendringer, register-endringer og nyinstallert programvare lagret i denne "sandboxen".
- Container Image: Når man har modifisert en container kan det hende at man ønsker å lagre den aktuelle sandboxen som et image. Dette imaget arver alle egenskapene fra sandboxen og tillater at nye containere kan baseres på denne. Dersom man for eksempel installerer en web server på en container, kan man lage et image av denne containeren og opprette flere containere basert på denne (web server forhåndsinstallert). Dette imaget inneholder endringene som ble gjort på den opprinnelige containeren og blir som et lag oppå det opprinnelige OS-imaget.
- Container OS Image: Det første og viktigste laget som til sammen blir en ferdig konfigurert container. Imaget inneholder operativsystemet og kan ikke endres. Endringer blir lagt i flere "lag" oppå dette imaget.
- Container Repository: Hver gang et container image blir laget blir dette imaget samt alle nødvendige ressurser lagret i et lokalt repository på container hosten. Imagene som er lagret her kan brukes av container hosten til å lage flere containere.

(Liste hentet fra [16])

3.1.1 Patching

Base imaget må ha den samme versjonen og patch nivået som host-operativsystemet den kjøres på. Hvis dette ikke stemmer vil det føre til ustabilitet eller at containere ikke starter opp [20] [21]. Ved eventuell patching må gamle containerene fjernes og nye containere må opprettes etter at et nytt container image er installert. Dette kan da føre til nedetid på serveren [22].

3.1.2 The Docker Hub

Docker Hub er en skybasert registertjeneste for å lagre og distribuere applikasjoner eller tjenester [23]. Det er en sentral ressurs for brukere av Docker der de kan oppdage nye container image, distribuere, og lage repository som man kan dele med andre brukere.

Docker Hub har følgende hovedfunksjonalitet:

- Image Repositories: Oppdage, styre, levere og hente image fra fellesskapet, offisielle eller private image biblioteker.
- Automatiserte Builds: Lager nye image automatisk når det blir registrert en forandring i et repository på GitHub eller BitBucket repository.
- Webhooks: En egenskap til Automated Builds, Webhooks lar deg sette i gang en handling etter en suksessfull opplasting til et repository.
- Organisasjoner: Lage arbeidsgrupper slik at kollegaer kan få tilgang til image repositories.
- GitHub og BitBucket integrasjon: Hent Docker image fra Docker til dine nåværende repositories.

(Liste hentet fra [23])

Du kan konfigurere Docker Hub repositories på to måter: Repositories lar deg levere image fra din lokale Docker daemon til huben. Automated Builds lar deg konfigurere

GitHub eller BitBucket til å trigge Docker Hub til å rebygge repositories når forandringer blir gjort på et repository.

Windows Server 2016 Technical Preview 5 ble tilgjengelig 27. April 2016. I den nye utgaven er det mulig å laste opp og hente image fra Docker Hub [24].

3.2 Containere på Windows Server 2016 - Technical preview 4

Denne rapporten baserer seg i stor grad på Windows Server 2016 - Technical Preview 4, da dette var den nyeste versjonen tilgjengelig ved arbeidsstart. Technical Preview 4 introduserte blant annet Hyper-V Containere med en høyere grad av isolering [25].

Med denne versjonen introduserte Microsoft to script. Ett som setter opp den fysiske maskinen som container host (vedlegg D), og ett som setter opp et Hyper-V virtuell maskin som container host. Gruppen startet med å sette opp den fysiske maskinen som container host, men dette var preget av feil. Etter en Windows-oppdatering ble versjonsnummeret til kernel feil, slik at container-imaget ikke ble godkjent, og containerne fikk derfor ikke startet. Dette problemet oppstod ikke med det andre scriptet, og det ble derfor bestemt å bruke en Hyper-V virtuell maskin som container host.

Microsoft introduserte også oppstartsveiledning for bruk av Powershell-kommandoer sammen med containere og en oppstartsguide for bruk av Docker-kommandoer med containere [26]. Docker-kommandoene varierte mellom å fungere og å ikke fungere (noe som kan forventes i en technical preview), og valget falt derfor på bruk av Powershell-kommandoer som virket mer stabilt (PowerShell og Docker ble presentert som likeverdige løsninger av Microsoft [26]).

For denne oppgavens del er det verdt å nevne at Hyper-V containere ikke fungerte i testmiljøet som ble satt opp. Dette kan ha bakgrunn i den noe tvetydige feilmeldingen markert i gult i figur 5. Det ble gjort forsøk på å løse problemet, ved reinnstallering av serveren, og oppsett av nye container hosts, men det ble dette arbeidet til slutt avsluttet. Det er derfor ingen praktisk del som inkluderer Hyper-V Containere i denne rapporten, og all informasjon om dette kommer fra andre kilder.

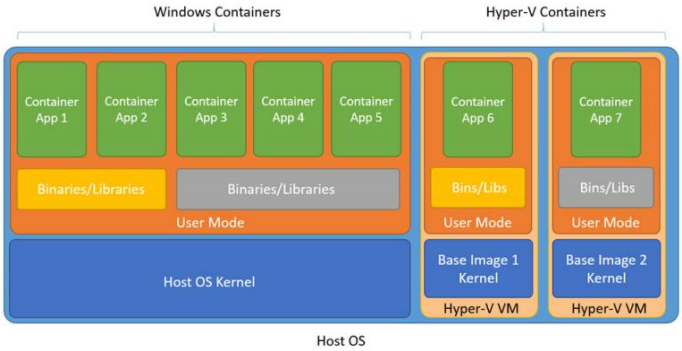
3.3 Windows Container vs. Hyper-V Container

I Windows Server 2016 Technical Preview 3 ble det implementert containere [27], som er integrert med Docker sitt repository og management system. Windows Server 2016 kommer med to type containere disse er, Windows Container og Hyper-V Container. Linux containere trenger Linux sitt grensesnitt fra host kernel og Windows Container trenger Windows sitt grensesnitt fra en host Windows Kernel, dermed kan ikke Linux containere kjøres på en Windows Server host eller en Windows Server container på en Linux host [28].

Windows Container deler kernel som fører til rask oppstartstid. Dette blir demonstrert i tabell 4. Containere deler operativsystem med hosten og med de andre kjørende containerene. Når mye data deles mellom containeren og hosten kan det føre til at en applikasjon får tilgang utenfor sin container og skader hosten eller andre containere, enten på grunn av feil i designet eller at namespace isoleringen er implementert feil, noe som kan føre til at systemet går ned. Windows Container er en løsning der operativsystemet stoler på applikasjonene som blir hostet på den, og alle applikasjonene som kjøres stoler på hverandre [17].

Det er noen utfordringer ved å bruke denne typen container i noen miljøer. For eksempel kan lav grad av isolering være et problem, ettersom isolasjonen skjer i user mode, som vist i figur 2. Dette vil si at containerene har en felles kernel. Dette vil ikke være noe problem i et enbrukermiljø der applikasjonene kan stoles på, men i et flerbrukermiljø kan andre brukere bruke felles kernel til å angripe andre containere. Containere er avhengig av versjon og patch-nivå av operativsystemet til hosten og det kan oppstå problemer hvis en patch som ødelegger for en applikasjon blir installert.

Hyper-V Containere er bygd opp på en litt annen måte enn Windows Container. Det er en mer isolert container, Hyper-V Containere har sin egen kopi av Windows kernel og har minne tildelt direkte til dem, dette fører til en god isolasjon [17]. De leverer samme type isolasjon som en vanlig Virtuell Maskin. Hosten blir bare eksponert for en liten del av grensesnittet til kommunikasjon og deling av hosten sine ressurser. På grunn av den sterke isoleringen har den ikke de samme ytelsesfordelene som en ordinær container, men kan fortsatt startes fra en dockerfil og automatiseres på lik linje med andre containere.



Figur 2: Vanlige containere, til venstre, deler på ressursene, mens Hyper-V containere, som vist til høyre, starter en egen virtuell maskin per container [28].

3.4 Fordeler og ulemper med containere

Selv om containere er allsidige og kan brukes til å kjøre mange applikasjoner er det ikke alltid at dette er det riktige valget for virtualisering. Et bruksområde som containere passer godt til, er utvikling av applikasjoner som består av flere programvarekomponenter. Hver komponent vil være plassert i en container og sammen vil containerene være koblet sammen, slik at de binder all programvaren sammen til en fungerende applikasjon [29]. Applikasjonens funksjonalitet kan da bli skalert og testet ved å legget til flere containere av en type komponent istedenfor en hel applikasjon.

Monolittiske applikasjoner som er selvforsynt med alle programvarekomponenter i en applikasjon, er ofte større og tyngre, og har ikke muligheten til å skalere opp en programvarekomponent som trenger mer ressurser [30]. Dersom applikasjonen er monolittisk må hel ny applikasjonen bli kjørt, selv om det bare en komponent er overbelastet og trenger mer ressurser.

3.4.1 Avhengigheter

En virtuell maskin kan kjøre et hvilket som helst operativsystem, mens en container bare kan kjøre samme operativsystem som hosten. For eksempel, Linux containere under Docker kan ikke kjøre Windows Servere eller motsatt [31].

3.4.2 Svakere isolasjon

Hypervisor-baserte virtuelle maskiner vil ha en høy grad av isolasjon fra hverandre siden ressursene til maskinvaren er virtualisert og presentert til de virtuelle maskinene gjennom hypervisoren. Det vil si at om en virtuell maskin blir rammet av feil, virus eller innbrudd vil dette ikke påvirke andre virtuelle maskiner. For containere vil hostens kernel være en "single point of failure", hvis hosten kræsjer, vil den også ta ned alle kjørende containere.

En angriper som klarer å få tilgang til en container burde under ingen omstendigheter få tilgang til andre container [32]. Som standard er det ikke namespace for brukere, slik at prosesser som kommer seg ut av en container vil ha samme rettigheter på hosten som den hadde i containeren. Om en prosess kjøres som administrator vil den også være administrator på hosten. Det er derfor viktig å være oppmerksom på angrep hvor en bruker potensielt klarer å tilegne seg samme rettigheter som en administrator gjennom for eksempel gjennom usikker applikasjonkode som ber om ekstra privilegier.

3.4.3 Styring av livssyklus

Containere kan opprettes og dupliseres raskt. Dette er en sentral egenskap for containere, men dette gir også mulighet til å konsumere en høy andel dataressurser dersom de ikke. Dette gjelder særlig for applikasjoner som skal skaleres, for eksempel web servere. Det koster mye å skalere opp unødvendig og da vil det være viktig å stoppe, eller slette containere når det ikke lenger er bruk for dem [33].

3.4.4 Verktøy

Containerteknologien er forholdsvis ny, og det mangler fortsatt modne verktøy til å overvåke og administrere containere på Windows. Det er noen verktøy som er tilgjengelige for containere [34], blant annet Kubernetes som er et open source verktøy til administrasjon, DockerUI som erstatter Linux kommandolinje med et web-basert grensesnitt og Logspout som kan rute containerlogger til en sentral lokasjon, men da primært for Linux.

Det manglet også gode verktøy for hypervisor-basert virtualisering i da teknologien var ny, men nå finnes det velutviklet verktøy som for eksempel OpenStack for administrasjon av sky og Foreman til overvåking av ytelse.

Magnum - Containers-as-a-Service

Magnum fra OpenStack kombinerer OpenStack, Docker Swarm, Kubernetes, Mesos og Flannel til å lage et Containers-as-a-Service-miljø [35]. Denne kombinasjonen gjør det mulig å kjøre og administrere containere ved bruk av OpenStack. Magnum er ikke en ny type teknologi som skal konkurrere med containere. Det er laget for å sørge for at nåværende containerteknologi og verktøy fungerer ved bruk av OpenStack [36]. Magnum er beregnet å kjøre minimalistiske host operativsystem som Fedora Atmoic, CoreOS eller Ubuntu Snappy. Ved kjøring av disse operativsystemene er det inkludert verktøy som gjør det mulig å kjøre Docker, Kubernetes og lignende [37].

Magnum har flere forskjellige type objekter i sitt system:

- Bay: En samling av node-objekter, en bay kan holde flere noder/containere samtidig.
- Node: En node er en-en relaterte Nova-instanser som er medlem av en bay.
- BayModel: Baymodel er en mal for å lage en bay, man kan liste ut de forskjellige malene som er tilgjengelige.
- Pod: En oversikt over antall kjørende containere på en fysisk eller virtuell maskin.
- Service:
- ReplicationController: ReplicationController brukes til å administrere en gruppe pods som gjør det mulig å sikre antall kjørende ressurser.
- Container: En Docker container.

(liste hentet fra [38])

3.4.5 Shipping

Utfordringen ved å shippe programvare er en av de store problemene for moderne IT-bedrifter. Denne utfordringen går ut på at det er et skille mellom utviklere og driftere i hvordan de prioritere ulike aspekter ved egenskaper og distribuering av en applikasjon som skal fungere i et produksjonsmiljø. Programvareutviklere må levere kode i henhold til sin kravspesifikasjon til driftere som må skape et kompatibelt miljø som koden kan kjøre i.

Det kan være vanskelig å flytte kode fra et miljø til et annet [34]. For at en applikasjon skal kjøre trenger den en rekke systemavhengigheter. Dette kan være operativsystem, middleware, bibliotek som for eksempel Visual C++ library eller annen programvare. Problemet vokser med antall avhengigheter, og vil kreve flere nedlastinger og mer diskplass. Det kan være tungvint og arbeidskrevende å finne frem alle nødvendige avhengigheter.I Docker kan utviklerne bygge en container med alle avhengighetene som applikasjonen trenger slik at de som jobber med drift kan kjøre applikasjonen uten å måtte ta hensyn til eksterne avhengigheter.

Et av de største problemene innenfor shipping, er oppdateringer som inneholder feil som forårsaker at funksjonalitet som tidligere fungerte, ikke lenger fungerer, såkalt regresjon [39]. Dette er svært skadelig for omdømmet til organisasjonen som publiserte koden. Kundene som benytter applikasjonen vil få nedsatt eller stanset produktivitet, kunne lide datatap, eller i svært kritiske applikasjoner, for eksempel medisinske journalsystem, føre til fare for menneskeliv. I Docker kan man rulle tilbake til et image hvor applikasjonen ikke viste tegn til problemer [40]. Docker containere kan starte opp igjen med en tidligere versjon av et applikasjonsimage raskere kontra virtuelle maskiner, noe som medfører lavere nedetid.

Det finnes flere problemer med shipping, men disse ble ansett som de mest aktuelle i forbindelse med Docker.

DevOps er en kultur hvor man legger vekt på samarbeid mellom utviklere og drift/annet IT-personell [41]. Målet med DevOps er å forenkle overgangen fra produktutvikling til drifting av det aktuelle produktet. Dette blir blant annet gjort for å øke både kvaliteten på sluttproduktet og hvor raskt det blir ferdig utviklet og satt i produksjon.

3.4.6 Oppgradering uten restart/stopp

En fordel med Docker er at det er mulig å oppgradere hosten uten å måtte restarte containeren. Dersom det blir oppdaget et sikkerhetshull i kernel, vil kernelen oppdateres og operativsystemet restartes. Noen ganger kan man bruke et system (som for eksempel Ksplice [42]) , som tillater oppgradering av virtuelle maskiner uten reboot. Uansett må man oppgradere til den nye kernelen og oppdatere alle virtuelle maskiner. Dette er lettere med containere, siden kernelen ikke er en del av et container image, så når hosten er oppdatert er også containeren oppdatert [43]. Dette er en stor fordel i forhold til virtuelle maskiner. Det finnes systemer som gjør det mulig å flytte en kjørende container fra en maskin til en annen. Det gjør det mulig å oppnå uavbrutte operasjoner mellom kerneloppgraderinger [44].

KAPITTEL

4

PRAKTISK DEMONSTRASJON AV POWERSHELL-ADMINISTRERTE WINDOWSCONTAINERE

I denne delen blir det demonstrert oppsett av container host i Windows Server 2016 - Technical Preview 4 samt hvordan man kan sette opp en container fra et image. Det blir i tillegg vist hvordan man lager et image av en container med en applikasjon installert for deretter å sette opp flere containere basert på dette imaget. Demonstrasjonen blir gjort ved hjelp av PowerShell.

```
PS C:\Users\Administrator> New-VMSwitch -name Vmswitch -NetAdapterName Ethernet -AllowManagementOS 1

Name     SwitchType NetAdapterInterfaceDescription
----     ---------- ------------------------------
Vmswitch External   QLogic BCM5709C Gigabit Ethernet (NDIS VBD Client) #34
```

Figur 3: Opprettelse av en Virtuell Switch

Med kommandoen *New-VMSwitch* kan virtuelle maskiner kan kommunisere med hverandre, og samtidig kan de koble seg opp mot internett. Dette blir vist i figur 3.

New-VMSwitch: Kommandoen for å lage en VMswitch.
-Name: Definerer navnet til switchen.
-NetAdapterName: Hvilket fysisk adapter den skal koble opp mot.
-AllowManagementOS: Spesifiserer om management operativsystemet kan bruke den fysiske adapteren som er bundet til den eksterne virtuelle switchen.

```
PS C:\Users\Administrator> wget -uri https://aka.ms/tp4/New-ContainerHost -OutFile c:\New-ContainerHost.ps1
```

Figur 4: Nedlasting av script fra Microsoft

Som vist i figur 4 brukes *wget* til hente filer fra en gitt URL *-uri* spesifiserer hvor man skal hente filen fra. *-outfile* spesifiserer hvilken plassering filen skal ha og navnet på den. Her hentes det ned et script fra Microsoft som setter opp en virtuell maskin som fungerer som en container host.

```
PS C:\Users\Administrator> powershell.exe -NoProfile c:\New-ContainerHost.ps1 -VmName testcont -WindowsImage ServerDatacenterCore -Hyperv

cmdlet New-ContainerHost.ps1 at command pipeline position 1
Supply values for the following parameters:
Password: ************
Before installing and using the Windows Server Technical Preview 4 with Containers virtual machine you must:
    1. Review the license terms by navigating to this link: http://aka.ms/tp4/containerseula
    2. Print and retain a copy of the license terms for your records.
By downloading and using the Windows Server Technical Preview 4 with Containers virtual machine you agree to such license terms. Please confirm you have accepted and agree to the license
terms.
[N] No  [Y] Yes  [?] Help (default is "N"): y
Using VHD path C:\Users\Public\Documents\Hyper-V\Virtual Hard Disks
Using external switch Vmswitch
The latest ServerDatacenterCore ISO is already present on this system.
Mounting ISO...
Converting WIM to VHD...

Windows(R) Image to Virtual Hard Disk Converter for Windows(R) 10
Copyright (C) Microsoft Corporation.  All rights reserved.
Version 10.0.10586.0.amd64fre.th2_release.151029-1700

INFO   : Looking for the requested Windows image in the WIM file
INFO   : Image 3 selected (ServerDataCenterCore)...
INFO   : Creating sparse disk...
INFO   : Mounting VHD...
INFO   : Initializing disk...
INFO   : Creating single partition...
INFO   : Formatting windows volume...
INFO   : Windows path (G:) has been assigned.
INFO   : System volume location: G:
INFO   : Applying image to VHD. This could take a while...
INFO   : Image was applied successfully.
INFO   : Making image bootable...
INFO   : Fixing the Device ID in the BCD store on VHD...
INFO   : Drive is bootable.  Cleaning up...
INFO   : Dismounting VHD...
INFO   : Closing Windows image...
INFO   : Done.
Dismounting ISO...
Creating temporary VHDX for the Containers OS Image WIM...
Initializing disk...
Saving Container OS image (WindowsServerCore) version 10.0.10586.0 from OneGet to F: (this may take a few minutes)...
Dismounting VHD...
Creating VHD files for VM testcont...
VHD mount must be synchronized with other running instances of this script.  Waiting for exclusive access...
Mounting ServerDatacenterCore VHD for offline processing...
Enabling Containers feature on drive G...
Enabling Hyper-V feature on drive G...
Copying Docker into ServerDatacenterCore VHD...
Copying NSSM into ServerDatacenterCore VHD...
This script uses a third party tool: NSSM. For more information, see https://nssm.cc/usage
Downloading NSSM...
Extracting NSSM from archive...
Writing default unattend.xml...
Copying Install-ContainerHost.ps1 into ServerDatacenterCore VHD...
Dismounting VHD...
Creating VM testcont...
Configuring VM testcont...
Connecting VM to switch Vmswitch
WARNING: Nested virtualization is an unsupported preview feature. Hypervisors other than the Hyper-V hypervisor running in a guest virtual machine are likely to fail. Furthermore, some
Hyper-V features are incompatible with nested virtualization, such as dynamic memory, checkpoints, and save/restore.
Starting VM testcont...
Waiting for VM testcont to boot...
Connected to VM testcont Heartbeat IC.
Waiting for specialization to complete (this may take a few minutes)...
Executing Install-ContainerHost.ps1 inside the VM...
Completing container install...
Querying status of Windows feature: Containers...
Feature Containers is already enabled.
Querying status of Windows feature: Hyper-V...
Feature Hyper-V is already enabled.
Waiting for Hyper-V Management...
Enabling container networking...
Creating container switch (NAT)...
Creating NAT for 172.16.0.0/24...
Installing Container OS image from D:\WindowsServerCore.wim (this may take a few minutes)...
```

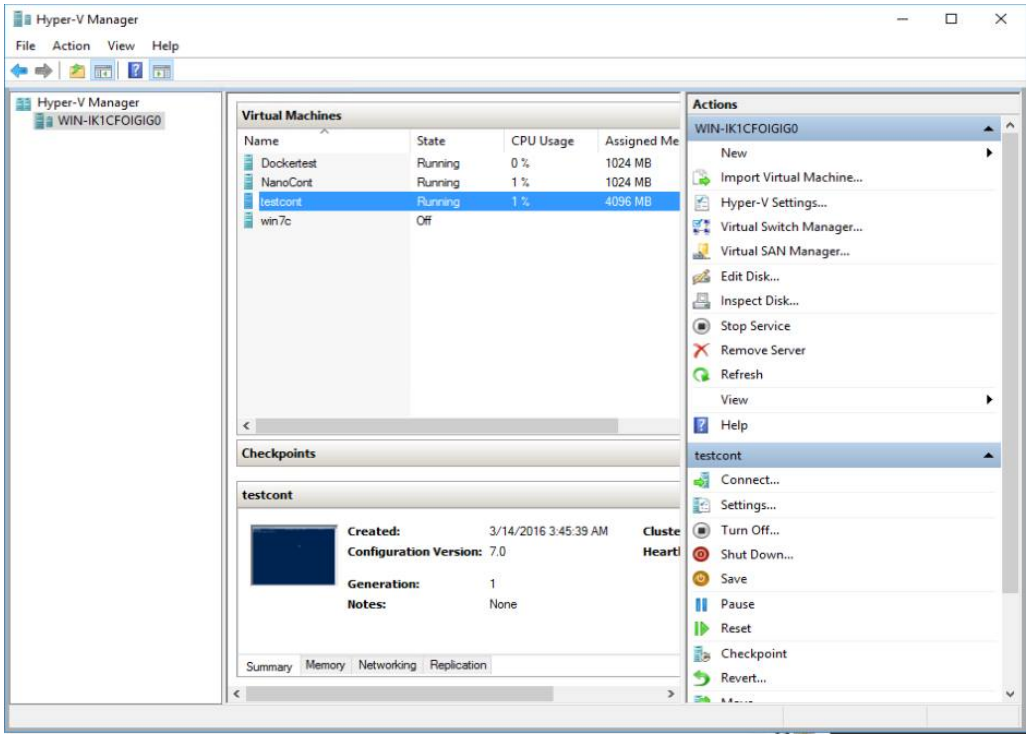
Figur 5: Opprettelse av container host

Scriptet fra Microsoft laster ned et operativsystem-image med containere aktivert og installerer det på en virtuell maskin (figur 5). Dette steget tar lang tid. Scriptet ligger i sin helhet som vedlegg D.

```
PS C:\Users> Get-ContainerHost

Name            ContainerImageRepositoryLocation
----            --------------------------------
WIN-IK1CFOIGIG0 C:\ProgramData\Microsoft\Windows\Hyper-V\Container Image Store
```

Figur 6: Det er opprettet en container host, og den ser etter container-image i mappen som er vist over

Get-ContainerHost viser at container hosten finnes og hvor den er lokalisert på datamaskinen som vist i figur 6.



Figur 7: Oversikt over tilgjengelige virtuelle maskiner i Hyper-V Manager

Figur 7 viser oversikten over tilgjengelige virtuelle maskiner ved bruk av Hyper-V manager. testcont er den som blir brukt i denne demonstrasjonen. Den har blitt tildelt 4GB minne, ettersom container-kommandoene var svært ustabile når hosten hadde mindre enn dette.

```
PS C:\Users\Administrator> Get-ContainerImage

Name              Publisher    Version      IsOSImage
----              ---------    -------      ---------
NanoServer        CN=Microsoft 10.0.10586.0 True
WindowsServerCore CN=Microsoft 10.0.10586.0 True
```

Figur 8: Oversikt over container images som er tilgjengelig.

Get-ContainerImage viser hvilke container-images som er tilgjengelige i container-hosten. Figur 8 viser at testsystemet hadde to alternativ, *Nanoserver* og *WindowsServer-Core*. Nanoserver er et nytt operativsystem i Windows Server 2016, som leveres helt uten GUI. I dette har Microsoft fjernet mange av de vanlige systemfilene som finnes i Windows Server Core. Denne styres via et eksternt verktøy, eksempelvis Windows PowerShell Remoting ettersom det ikke er mulig å logge inn på denne serveren. Windows Server Core er et operativsystem med GUI, men uten skrivebordstøtte og Windows Explorer.

Denne kan logges inn på, blant annet ved bruk av *Enter-PsSession* eller remote desktop [45].

```
PS C:\Users\Administrator> Start-Container nanotest
Start-Container : 'NanoTest' failed to start.
'NanoTest' failed to initialize: The operating system of the container does not match the operating system of the host. (0xC0370101).
'NanoTest' failed to start. (Container ID FFDA0F96-E879-41F2-B2A9-80A5A643BAF6)
'NanoTest' failed to initialize: The operating system of the container does not match the operating system of the host. (0xC0370101). (Container ID
FFDA0F96-E879-41F2-B2A9-80A5A643BAF6)
At line:1 char:1
+ Start-Container nanotest
+ ~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (:) [Start-Container], VirtualizationException
    + FullyQualifiedErrorId : OperationFailed,Microsoft.Containers.PowerShell.Cmdlets.StartContainer
```

Figur 9: Nanoserver fungerer ikke på en host som kjører windowsservercore

I motsetning til på Linux, hvor Docker lager en virtuell maskin for å kjøre en container av et annet operativsystem enn hosten, er det kun mulig å lage containere med samme operativsystem som hosten har. Som feilmeldingen i figur 9 viser.

```
PS C:\Users\Administrator> Start-Process powershell -Verb runas
```

Figur 10: Kommando som brukes til å starte PowerShell i et nytt vindu med administratorrettigheter.

Start-Process kommandoen starter Powershell i nytt vindu med administratorrettigheter (figur 10).

```
PS C:\Users\Administrator> New-Container -Name Containerdemo -ContainerImageName WindowsServerCore -SwitchName "Virtual Switch"

Name          State Uptime   ParentImageName
----          ----- ------   ---------------
Containerdemo Off   00:00:00 WindowsServerCore


PS C:\Users\Administrator> Get-Container

Name          State Uptime   ParentImageName
----          ----- ------   ---------------
Containerdemo Off   00:00:00 WindowsServerCore


PS C:\Users\Administrator> Start-Container ContainerDemo
```

Figur 11: Oversikt over container images som finnes og hvilke containere som er kjørende.

For å få en oversikt over tilgjengelige images kjøres kommandoen *Get-ContainerImage*. Ettersom containeren skal benytte WindowsServerCore oppretter en containeren basert på det aktuelle imaget med kommandoen: *New-Container -Name* ***valgfrittnavn*** *-ContainerImageName* ***navnpåimage*** *-SwitchName* ***navnpåswitch***. Dersom kommandoen ble gjennomført korrekt vil en få frem en liste over tilgjengelige containere. Den nye containeren blir startet med *Start-Container* ***navnpåcontainer*** (figur 11).

```
PS C:\Users\Administrator> Enter-PSSession -ContainerName ContainerDemo -RunAsAdministrator
[ContainerDemo]: PS C:\Windows\system32> Add-WindowsFeature Web-Server

Success Restart Needed Exit Code      Feature Result
------- -------------- ---------      --------------
True    No             Success        {Common HTTP Features, Default Document, D...
```

Figur 12: Logger seg inn på containeren ContainerDemo, deretter installeres Web-Server på denne containeren.

Etter at denne nye containeren er opprettet skal det installeres en web-server på den. For å logge seg inn på en container brukes *Enter-PsSession -ContainerName* ***navnpåcontainer*** *-RunAsAdministrator*. Her installeres nødvendig programvare, i dette tilfelle web-server. Det blir gjort ved hjelp av *Add-WindowsFeature Web-Server* og avslutter med å forlate containeren med *exit*.

Dette er nå en fungerende container med web-server installert. Det neste steget vil være å lage et container-image basert på denne (figur 12).

```
PS C:\Users\Administrator> New-ContainerImage -ContainerName ContainerDemo -Name IISContainer -Versi
on 1.0 -Publisher NTNU

Name         Publisher Version IsOSImage
----         --------- ------- ---------
IISContainer CN=NTNU   1.0.0.0 False
```

Figur 13: Lager ett nytt container-image som skal inkludere endringer som er blitt gjort på containeren ContainerDemo.

Det nye imaget skal inkludere endringene som ble gjort på containeren, altså webserveren. Dette blir gjort med *New-ContainerImage -ContainerName* ***navnpåcontainer*** *-Name* ***navnpånyttimage*** *-Version* ***versjonsnr*** *-Publisher* ***navnpåbedrift*** (figur 13).

Som vist i figur 13 er *IsOSImage* satt til false på det nye imaget, da dette imaget bruker windowsservercore som OS-image. På dette tidspunktet kan en fjerne container som ble brukt for å skape container-imaget. Dette gjøres med *Remove-Container* ***navnpåcontainer***.

Det nye imaget er klart til bruk, dersom en ønsker å sette opp ti identiske web-servere kan det gjøres på følgende måte :

```
for($i=1; $i -le 10; $i++)
{
        New-Container -Name IIS$i -ContainerImageName IISContainer
        -ContainerComputerName Container-$i -Switchname "Virtuell switch"
}
```

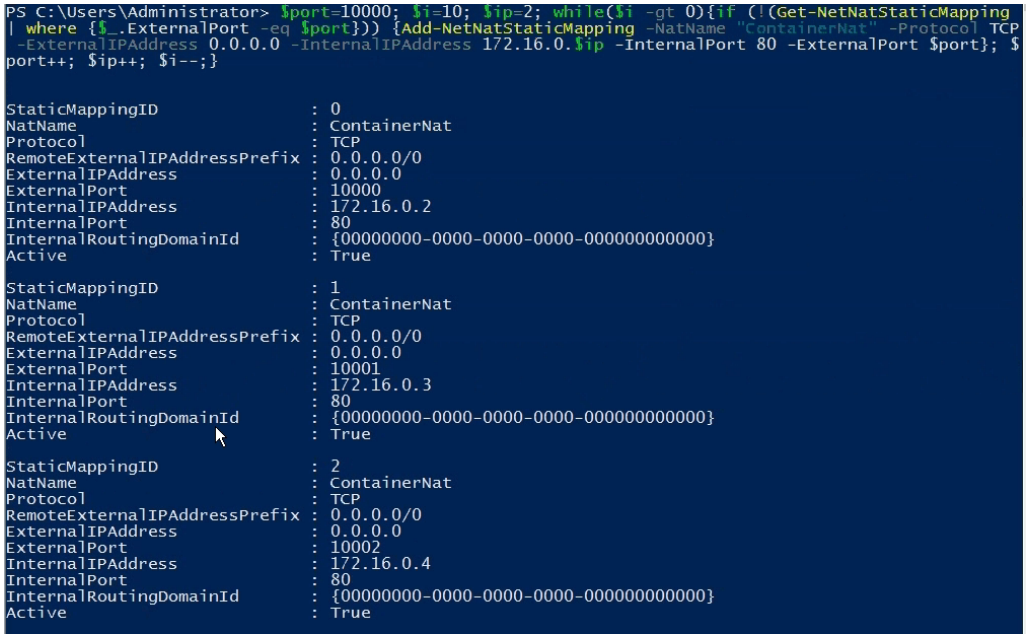
```
PS C:\Users\Administrator> for($i=1; $i -le 10; $i++) {New-Container -Name IIS$i -ContainerImageName
 IISContainer -ContainerComputerName Container-$i -SwitchName "Virtual Switch"}

Name  State Uptime   ParentImageName
----  ----- ------   ---------------
IIS1  Off   00:00:00 IISContainer
IIS2  Off   00:00:00 IISContainer
IIS3  Off   00:00:00 IISContainer
IIS4  Off   00:00:00 IISContainer
IIS5  Off   00:00:00 IISContainer
IIS6  Off   00:00:00 IISContainer
IIS7  Off   00:00:00 IISContainer
IIS8  Off   00:00:00 IISContainer
IIS9  Off   00:00:00 IISContainer
IIS10 Off   00:00:00 IISContainer
```

Figur 14: Ti containere blir opprettet basert på IIS-imaget.

Som figur 14 viser blir det laget ti containere, alle basert på imaget som ble laget i figur 13. Containerne er for øyeblikk skrudd av. Følgende kommando skrur på samtlige containere: *Get-Container | Sort-Object -Property name | Start-Container*. Det sorteres på containernavn for å sørge for at containerene starter i riktig rekkefølge.

Videre ønsker vi å sette opp unike porter og gjøre portmapping slik at det blir mappet en port til containerens egen port 80 for alle for de ulike web serverne slik at de kan nås i en nettleser (figur 15). Det er satt opp ti containere og dersom en begynner på portnummer 10000 vil kommandoen i dette tilfelle bli:

```
$port=10000;
$i=10;
$ip=2;
while($i -gt 0)
{
  if (!(Get-NetNatStaticMapping | where {$_.ExternalPort -eq $port}))
  {
     Add-NetNatStaticMapping -NatName "valgfrittnavn"
     -Protocol TCP -ExternalIPAddress 0.0.0.0
     -InternalIPAddress 172.16.0.$ip
     -InternalPort 80 -ExternalPort $port
  };
  $port++;
  $ip++;
  $i--;
}
```



Figur 15: Mapping av porter fra en intern port til en ekstern port.

Det er i tillegg nødvendig å lage en brannmurregel for å tillate tilgang til web serverne.

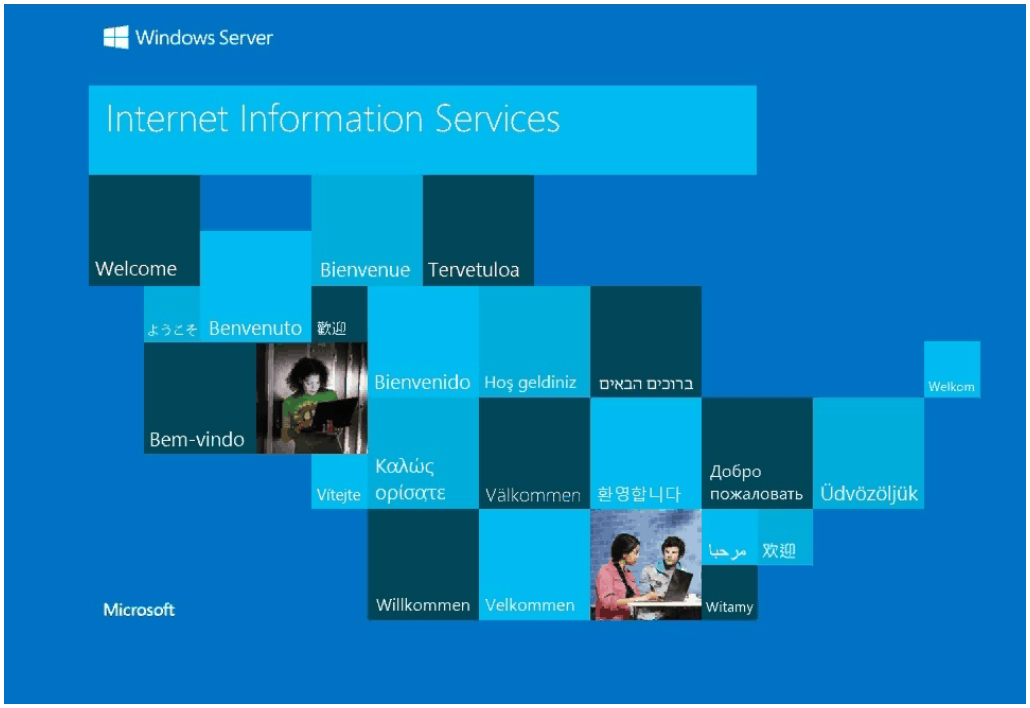
Dette kan gjøres gjøres ved hjelp av Desired State Configuration (DSC), på følgende måte:

```
configuration AddFirewallRule
{
  Import-DscResource -ModuleName xNetworking

  Node "localhost"
  {
    xFirewall Firewall
    {
      Name = "TCP80"
      DisplayName = "HTTP on TCP/80"
      Ensure = "Present"
      Enabled = "True"
      LocalPort = ("80")
      Protocol = "TCP"
      Description = "Brannmurregel for webservere"
    }
  }
}

AddFirewallrule
Start-DscConfiguration -path AddFirewallRule -Wait -Verbose -Force
```

Install-Module xNetworking [46]må kjøres i forkant dersom modulen ikke allerede er installert på container host.

Alternativt kan man opprette brannmurregelen uten DSC:

```
New-NetFirewallRule -Name "TCP80" -DisplayName "HTTP on TCP/80" -Protocol
tcp -LocalPort 80 -Action Allow -Enabled True
```



Figur 16: Standardsiden til Internet Information Services (IIS).

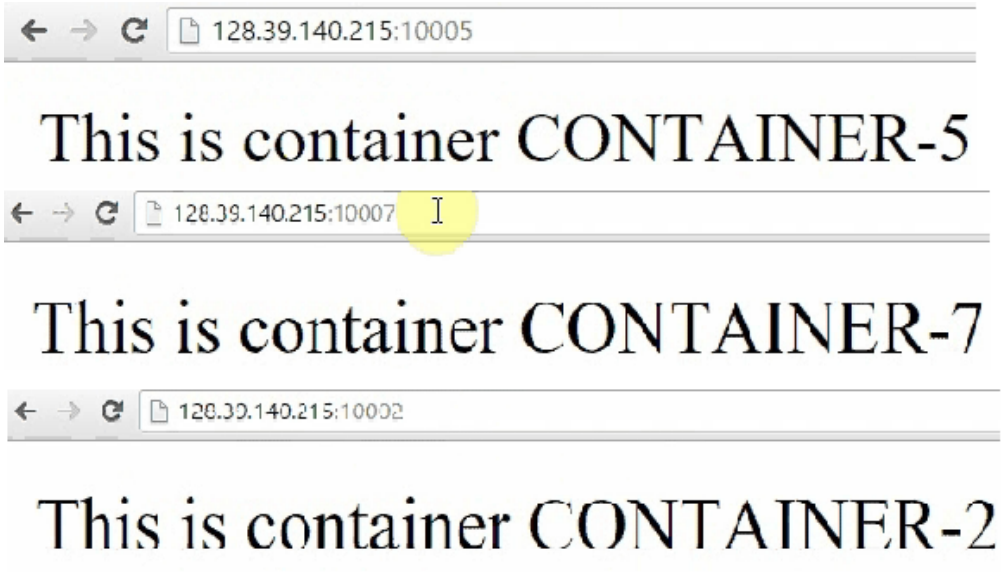
De ulike web serverne kan nå nås via en nettleser, men de viser alle den samme siden som vises i figur 16. For å demonstrere at det er flere ulike web servere ble de endret slik at de hadde unike nettsider. *Invoke-Command* brukes for å endre på en container uten å gå inn på den aktuelle containeren, som figur 17 viser. Dette gjøres kan enkelt demonstreres ved å la hver container vise navnet sitt:

```
for ($i=1; $i -le 10; $i++) {
        Invoke-Command -ContainerName iis$i -RunAsAdministrator {
                Remove-Item C:\inetpub\wwwroot\iisstart.htm;
                "This is container $env:COMPUTERNAME" >
                C:\inetpub\wwwroot\index.html
        }
}
```

```
PS C:\Users\Adminstrator> for ($i=1; $i -le 10; $i++) {Invoke-Command -ContainerName iis$i -RunAsAdministrator
{Remove-Item C:\inetpub\wwwroot\iisstart.htm;"This is container $env:COMPUTERNAME" > C:\inetpub\wwwroot\index.h
tml}}
```

Figur 17: *Invoke-Command* brukes for å sende kommandoer til eksterne maskiner, i dette tilfelle containeren.

Resultatet blir som vist i figur 18. IP-adressen samt portnummer vil gi direkte tilgang til containeren en ønsker å aksessere.



Figur 18: Resultat som viser at nettsiden er tilgjengelig og hvilken container som holder denne siden i drift.

KAPITTEL

5

CASE

Sammen med oppdragsgiver og veileder ble det utarbeidet to caser som skulle utføres. På grunn av tekniske årsaker kunne ingen av disse to gjennomføres. Dette blir nærmere forklart i kapittel 5.1.2. Ettersom disse casene ikke kunne fullføres, ble det undersøkt alternative praktiske demonstrasjoner av teknologien. Her ble det blant annet sett på dynamisk skalering av web servere, oppsett av andre typer servere. Disse ble forkastet på grunn av manglende funksjonalitet, kompleksitet, manglende nytte for Ikomm, og manglende kilder til informasjon om Windows Container. Til slutt ble det bestemt at vi skulle se nærmere på ytelsen til containere i Windows Server 2016. Dette er nærmere forklart i kapittel 5.3

5.1 Case 1

5.1.1 Hva er Virtual Desktop Infrastructure (VDI) og terminal server

VDI er et skrivebord som kjører i en virtuell maskin på server i et datasenter, og som tillater personaliserte skrivebord for hver bruker [47]. Det finnes flere modeller av skrivebordsvirtualisering, som blir inndelt i to kategorier, basert på om operativsystemet blir kjørt på den lokale maskinen eller sentralt på serveren. Skrivebordsvirtualisering blir kjørt fra et image på serveren og all data blir lagret på der og ikke på brukerens datamaskin [48]. Noe som både er en sikkerhetsfordel, ettersom dataen ikke er fordelt utover mange datamaskiner, men betyr også at serverne som hoster skrivebordene er et single point of failure [49].

En terminalserver er en datamaskin eller en server som gir ut terminaler til datamaskiner, skrivere eller andre enheter, og som kobler seg opp på et felles lokalt nettverk eller over et trådløst nettverk. Historisk sett kobler terminalene seg opp til en terminal server ved å bruke RS-232C eller RS-423 port, som er en treg tilkobling mellom datamaskinene og andre enheter. Den andre siden av terminal-serveren er koblet til nettverket med integrert nettverkskort som er koblet opp mot det lokale nettverket vanligvis med en nettverkskabel [50].

5.1.2 Gjennomføring

Den første casen gikk ut på å sette opp en VDI løsning ved bruk av containere, for å se om dette var noe Ikomm kunne ta i bruk i sine systemer. Dette er ikke støttet i gjeldende versjon av Windows Server 2016 (Technical preview 4) [21], og ifølge Taylor Brown vil dette heller ikke bli støttet når den blir lansert [2].

> We do not have RDP, we had it in TP3. We don't have it in TP4, I don't belive we will have it in RTM, the reason we had it was for GUI based applications so we could click next and go through them, it completely breaks the Docker build experience. We're really pushing automated ways to do it. So we can get the full experience around it. Now over time I can see some interesting use cases for client based and GUI based container. For right now we will start with server containers.
> - Taylor Brown [2]

Denne casen blir derfor ikke gjennomførbar.

5.2 Case 2

> «Når det gjelder oppgaven og «Case 2» så har vi hatt et internt møte og kommet opp med forslag til applikasjon som kan være en case: Huldt og Lillevik. http://www.huldt-lillevik.no/lonn/ Dette er en applikasjon noen av kundene våre benytter, og er en

> utbredt applikasjon her på berget, og kan være aktuell å kjøre for flere kunder via Docker. Applikasjonen består av en baseknytning, en «fagapplikasjon», samt en klient som da er aktuell å kjøre på VDI. Vi benytter HC Dahl - http://www.hcdahl.no/ som ressurs når det gjelder Huldt og Lillevik, så dere kan sikkert gjøre en forespørsel til dem om bistand/avklaringer rundt applikasjonen.» - Mail fra Ikomm 16.02.2016

I case 2 skulle Huldt og Lillevik sitt lønnssystem bli satt opp. Dette er et system som består av en baseknytning, en "fagapplikasjon", samt en klient som hadde vært aktuelt å kjøre på VDI. Ettersom det ikke er mulig å få et grafisk brukergrensesnitt fra en container i Windows, ble dette caset forkastet, da administrasjon av denne applikasjonen krever grafisk brukergrensesnitt.

5.3 Alternativ til case

Grunnet problemer med casene, ble det etter intern diskusjon og gjennomgang med veileder bestemt at gruppen skulle se nærmere på ytelsestesting. I disse testene ble det målt CPU, minne, I/O og nettverksytelse. Det ble også gjort forsøk på å teste random access, men på grunn av manglende software for testing som kjører på Windows uten grafisk grensesnitt, ble ikke dette gjennomført. Vi valgte å utføre ytelsestester for å bedre kunne besvare spørsmålene fra Ikomm. Vi anser spesielt spørsmålene om gevinstrealisering og om Docker er en konkurrent til nåværende teknologi som relevante (se delkapittel 1.1 for oppgavebeskrivelse, og kapittel 8 for resultatene fra denne testingen).

KAPITTEL

6

DOCKER ARKITEKTUR

6.1 Linux container vs Windows Container - Hva er forskjellen?

Linux og Windows Container er veldig like, sett fra utsiden. Begge utnytter samme teknologi i kernel og i kjernen av operativsystemet. Forskjellen kommer tydelig frem i form av ulike plattformer og workloads i containerene. På en Windows Server Container kan en kjøre eksisterende Windows-teknologi, for eksempel .NET, ASP.NET og Powershell [51].

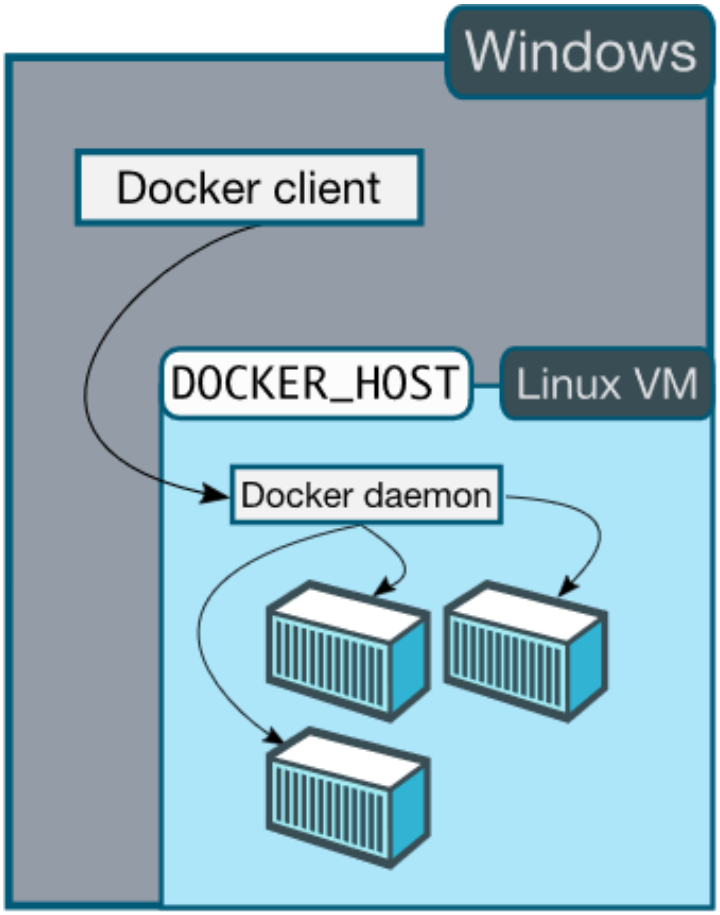
Windows Server Container støtter ikke Remote Desktop Protocol (RDP) i Technical Preview 4 og det ser ikke ut til at det blir støttet i retail Windows Server 2016 heller [2]. Dette medfører blant annet at VDI-løsninger ikke vil være støttet på Windows Server containere.

Docker containere og LXC (Linux Containere) containere har lignende sikkerhetsegenskaper. Når en container settes opp med *docker run* vil Docker lage et sett med namespace og kontrollgrupper for containeren. Det er dette som gjør det mulig å isolere prosesser uten bruk av virtuelle maskiner. Namespace er den sikkerhetsmekanismen som vil gi isolasjon til containere. Isolasjonen vil hindre at prosesser i containere kan se eller påvirke prosesser i andre containere, eller de prosessene som kjøres av hostsystemet [52].

6.2 Docker Engine

Docker Engine er kjernen i Dockers plattform, og er et verktøy som lager og kjører Docker-containere. Docker Engine (Docker daemon i figur 19) benytter Linux for å opprette miljøet som applikasjonene skal kjøre i. Klienten kommuniserer med Docker daemon for å opprette og kjøre containere [53].

6.2.1 Docker Client



Figur 19: Docker klienten kjører på windows, mens daemonen kjører i en virtuell maskin som kjører Linux [54].

Ved bruk av Docker client på Windows kjører Docker daemon i en Linux-basert virtuell maskin. Docker-klienten brukes til å få kontakt med Docker host, som gjør at man får kontakt med containerene som kjører inne i denne hosten (figur 19).

6.2.2 Docker Compose

Docker Compose er et verktøy som gjør det mulig å kjøre flere container servicer ved å konfigurere en Docker Compose-fil [55]. Man legger til konfigurasjonene i en YAMl fil, docker-compose.yml, der det spesifiserer hvilke images og konfigurasjoner man ønsker. For å kjøre denne filen brukes kommandoen *docker-compose up*. Denne kommandoen lager images fra dockerfiler, hente images fra registrene, lager og starter containerene, holder oversikt over loggene [56].

6.2.3 Docker Swarm

Docker Swarm er et grupperings- og planleggingsverktøy for Docker containere [57]. Med dette verktøyet kan utviklere og administratorer etablere og administrere et stort antall Docker hosts i et virtuelt system. Grupperinger som dette er en viktig del av

containerteknologien siden den lager disse gruppene av systemer som øker redundans hvis en eller flere noder/hosts feiler. Det er også mulig å legge til og fjerne container-iterasjoner, etter hvert som ressurskravene endres.

Swarm passer på at det distribueres nok ressurser til alle kjørende containere til en hver tid, ved at den automatisk optimaliserer ressursene til den hosten som har mest belastning. Dette gir en grunnleggende lastbalansering for container-applikasjoner, som sikrer at containere har tilstrekkelige ressurser, og samtidig opprettholder nødvendig ytelesesnivå [58].

En swarm kontrolleres via en swarm manager, med mulighet til å sette opp flere for redundans, som konfigurerer tidsplaner for containerne som er del i swarmen [59].

6.2.4 Docker Registry

Med Docker-registrene kan man lagre og distribuere Docker images. Registrene er open-source med tillatelse fra Apache lisensen.

6.3 Windows

6.3.1 Compute Services

Host compute service er et lav nivå Application Programming Interface (API) som er bygd på toppen av control groups, namespaces og layer capabilites [60]. Det er et felles sted som containeroperasjoner kan kommunisere med. Med denne API-en slipper man å programmere direkte mot de underliggende lagene og kan gi enkle kommandoer som for eksempel "lag en container".

6.3.2 Control groups

Job Objects

Et jobb-objekt brukes til å samle prosesser i en gruppe slik at de kan kontrolleres som en enhet, til å begrense ressurser for alle prosessmedlemmene, og vedlikeholde attributt-informasjon [61]. En operasjon på et jobb-objekt vil påvirke alle prosesser som er medlem av jobb-objektet. For eksempel å begrense størrelsen på *working set* og prosessprioritering eller å avslutte prosesser assosiert med en jobb.

For å lage et jobb-objekt kjøres kallet CreateJobObject som tar to argumenter, et navn og et sikkerhetsattributt. Dette vil returnere en jobb-objekt handle [62]. OpenJobObject kan åpne en eksisterende jobb som sendes med navnet til den aktuelle jobben. CloseHandle vil avslutte en jobb. AssignProcessToJobObjects legger til en prosess i et jobb-objekt. Det tar to argumenter. En prosess kan ikke være medlem av mer enn en jobb. Derfor vil funksjonen feile om en prosess allerede er assosiert med en annen jobb. Når en jobb er lagt til en jobb vil den arve alle begrensninger som er assosiert med jobben, og all informasjon angående attributter, for eksempel prossesstid.

SetInformationJobObject er en funksjon som kan begrense kontrollen til en jobb [63]. Den har en parameter JobObjectInfoClass som for eksempel kan bruke klassen JobObjectBasicInformationClass til å begrense tiden hver prosess får bruke i user-mode, tiden en jobb får bruke i user-mode, minimum og maksimum størrelsen på working set, antall aktive prosesser, prioritet, prosessordeltakelse (hvilken CPU-kjerne(r) en prosess skal kjøre på).

6.3.3 Namespace

Namespace Objects

Object namespace beskytter objekter med navn mot uautorisert tilgang [64]. Ved å lage private namespace økes sikkerheten for de applikasjonene som kjører i miljøet.

Networking

Windows Container fungerer på lignende måte i forhold til virtuelle maskiner i et virtuelt nettverk. Hver container har en virtuelt nettverksadapter som er koblet til en virtuell switch som forflytter inngående og utgående trafikk. For å forsterke isolasjon mellom containere på hosten blir et nettverksområde laget for Windows- og Hyper-V-Containere, der en nettverksadapter for containeren blir installert. Windows Containere bruker et vNIC fra hosten til å koble seg til den virtuelle switchen. Hyper-V Containere bruker et eget virtuelt NIC, som ikke er tilgjengelig for andre deler av systemet, til å koble seg til den virtuelle switchen.

Windows Container støtter fire nettverksmoduser:

- Network Address Translation Mode: Hver container er koblet til en intern virtuell switch, og vil bruke WinNat til å koble seg til en private IP subnet. WinNat vil ta seg av både Network Address Translation (NAT) og port address translation (PAT) mellom container hosten og andre containere.
- Transparent Mode: Hver container er koblet til en ekstern virtuell switch og vil være direkte tilkoblet til det fysiske nettverket. IP-adresser kan bli tildelt statisk eller dynamisk ved å bruke en ekstern DHCP server. Nettverkstrafikken fra containerne i form av frames blir plassert direkte på det fysiske nettverket uten adresseoversetting.
- L2 Bridge Mode: Hver container er tilkoblet en ekstern virtuell switch. Nettverkstrafikk mellom to containere i samme IP-subnet som er knyttet til samme host, vil være direkte koblet sammen i en bro. Nettverkstrafikk mellom to containere på forskjellige subnet eller som er knyttet til forskjellige container hoster vil bli sent ut gjennom den eksterne virtuelle switchen. Utgående nettverkstrafikk som kommer fra containere vil få source MAC adresse omskrevet til den samme som container hosten. Inngående nettverkstrafikk som går til en container vil få destinasjon MAC adresse omskrevet til sin egen.
- L2 Tunnel Mode: Denne modusen ligner på L2 Bridge mode, men er ment spesifikt for Microsoft Cloud Stack. Hver container er koblet til en ekstern virtuell switch der MAC adresser blir omskrevet på utgående og inngående trafikk. Forskjellen er at all nettverkstrafikk fra containere blir forflyttet til den fysiske hostens virtuelle switch uavhengig av Layer-2 tilkobling. Dette gjør at nettverk policy kan opprettholdes i den fysiske hostens virtuelle switch, f.eks. av Network Controller eller Network Resource Provider.

(Liste hentet fra [65])

Process table

Dersom man har 10 containere, vil hvilken som helst gitt container kun se sine egne prosesser da det er kun disse som er inkludert i containeren sitt namespace.

Nedenfor blir det demonstrert prosesskjuling på containere:

```
[contPidShow]: PS C:\Windows\system32> Get-Process

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id  SI ProcessName
-------  ------    -----      ----- -----   ------     --  -- -----------
     87       5      956       4572 ...72     0.08   5832   2 CExecSvc
    171       9     1636       3928 ...97     0.33  12252   2 csrss
    103       6      900       4684    49     0.02   9344   2 geekbench
    178      15    82368      80552   162   210.91   9424   2 geekbench_x86_32
      0       0        0          4     0                0   0 Idle
    676      19     3360      10248 ...96     0.33  12092   2 lsass
    469      24    51620      65460 ...51     1.22   7744   2 powershell
    602      25    55136      70212 ...58     1.63  13488   2 powershell
    202      10     2596       5808 ...76     0.44  12108   2 services
     46       3      376       1140 ...59     0.08  12320   0 smss
    289      13     2812      10136 ...34     0.31   3944   2 svchost
    957      33    13984      27880 ...17     2.45   5708   2 svchost
    273      14     6640      10928 ...94     0.47   5712   2 svchost
    391      83     6996      16396 ...05     2.17   6296   2 svchost
    226      15     3176       9476 ...98     0.25   6760   2 svchost
     81       6      884       4728 ...86     0.05   7380   2 svchost
    259      14     2168       6544 ...87     0.13   8832   2 svchost
    297      12     2832       8708 ...96     0.25  11900   2 svchost
   3263       0      124        144     3    67.50      4   0 System
```

Figur 20: prosesser kjørende i containeren “contPidShow”

I figur 20 vises kjørende prosesser på en container med navn “contPidShow”. Geekbench™ kjører på containeren med prosess-Id “9344”.

```
PS C:\> Get-Process

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id  SI ProcessName
-------  ------    -----      ----- -----   ------     --  -- -----------
     81       5      956       4572 ...72     0.09   5832   2 CExecSvc
     71       4      884       4464 ...71     0.06  17056   3 CExecSvc
     39       3     1536       2780 ...70     0.03   2792   1 cmd
    117      10     8940      10804 ...62    12.55   2800   1 conhost
    208      11     1740       4120 ...01     0.34    744   0 csrss
    138      10     1304       7484 ...01     3.64    828   1 csrss
    165       9     1636       3920 ...97     0.33  12252   2 csrss
    161       9     1276       3800 ...97     0.20  18600   3 csrss
    103       6      900       4684    49     0.02   9344   2 geekbench
    186      16    78420      80400   204   321.55   9424   2 geekbench_x86_32
```

Figur 21: Oversikt over kjørende prosesser på container host

På host vises Geekbench™ i listen over prosesser med samme prosess-Id som på “contPidShow”(Figur 21). Legg også merke til at den inneholder flere prosesser fra begge containerene i tillegg til sine egne unike prosesser.

```
[NoGeek]: PS C:\Windows\system32> Get-Process

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id  SI ProcessName
-------  ------    -----      ----- -----   ------     --  -- -----------
     79       5      952       4500 ...72     0.06  17056   3 CExecSvc
    149       9     1268       3772 ...96     0.19  18600   3 csrss
      0       0        0          4     0                0   0 Idle
    551      19     2860       9460 ...94     0.38  18440   3 lsass
    642      30    58156      75836 ...75    11.42  15728   3 powershell
    198      10     2748       5948 ...76     0.50  18456   3 services
     46       3      380       1140 ...59     0.11   8476   0 smss
     81       6      872       4728 ...86     0.05  17076   3 svchost
    384      30     4768      14632 ...78     0.50  17344   3 svchost
    226      15     3292       9612 ...98     0.28  17588   3 svchost
    944      34    10976      25972 ...95     1.92  17608   3 svchost
    275      14     6876      10896 ...94     0.44  17748   3 svchost
    288      13     2792      10140 ...34     0.33  17832   3 svchost
    244      14     2264       6468 ...87     0.20  18180   3 svchost
    291      12     2692       8540 ...95     0.23  18244   3 svchost
   3530       0      124        144     3    63.03      4   0 System
```

Figur 22: Prosesser kjørende i containeren “NoGeek”

En annen container med navn “NoGeek” kjører, men uten at Geekbench™ blir synliggjort i listen over prosesser selv om den kjører på “contPidShow” (figur 22).

```
PS C:\> Get-Container | Sort-Object -Property State

Name        State   Uptime          ParentImageName
----        -----   ------          ---------------
NoGeek      Running 00:07:51.3270000 WindowsServerCore
contPidShow Running 00:03:57.8310000 WindowsServerCore
```

Figur 23: Oversikt over kjørende containere

Som vist i figur 23 kjører både “NoGeek” og “contPidShow” på samme container host.

6.3.4 Layer Capabilities

Registry

Windows registeret er en hierarkisk database som brukes til å lagre informasjon og innstillinger for programvare, maskinvareenheter, brukerpreferanser, operativsystemets konfigurasjon og mye annet.

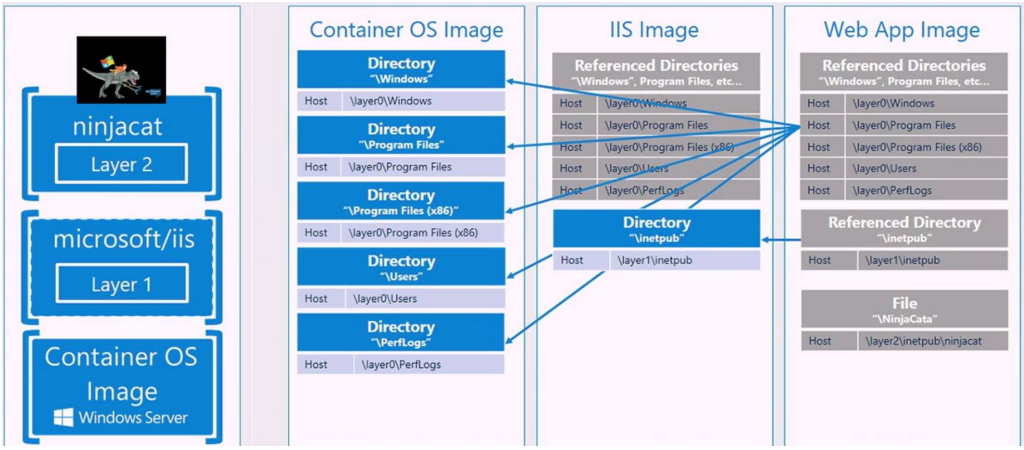
Union like filesystem extensions

Et union filsystem er et filsystem som slår sammen en samling av forskjellige filsystemer og kataloger til et logisk filsystem. Docker i Windows bruker file system filter interface for å skape virtuelle representasjoner for et sett med filer.

En container bygges opp av container image som er en filbasert template og er Read-

only [2]. I et image er det konfigurasjonsinnstillinger som Docker implementerer i en kombinasjon av metadata og lag. Image metadata inneholder slikt som navnet på et image, prosessinformasjon, ressursdeling og nettverksinnstillinger. Hvert lag består av metadata som forteller hva som er innholdet i laget og payload som inneholder filene og registerkonfigurasjon for laget.

Et containerOS-image er en ren kopi av Windows Server sine kataloger: *"\Windows"*, *"\Program Files"*, *"\Program Files (x86)"*, *"\Users"*, *"\PerfLogs"*. Når det blir opprettet en container vil OS image sine filer bli lagt til som en transparent representasjon i en container sandbox. Disse filene vil ikke eksistere som en kopi i sandbox men vil reflektere OS image ved hjelp av pekere.



Figur 24: Lagdelingen av et container-image [2]

Kataloger til ny programvare som blir installert vil havne i sandbox. Ved å installere Web Serveren IIS som blant annet inneholder katalogen "\inetpub" vil denne eksempelvis bli plassert i sandbox. Ved å gjøre en commit vil alle filene i sandbox bli flyttet over til et image. Det går nå an å lage en sandbox av dette IIS image og lage en nettside som for eksempel Ninja Cat som inneholder katalogen *"\NinjaCat"*. Igjen, ved å kjøre en commit, vil denne sandboxen flytte filene over i et image. Det er nå tre image, Container OS Image, IIS Image og Web App Image. IIS Image vil ha en katalog *"\inetpub"* foruten katalogreferanser til Container OS image. Web App Image vil ha en katalog *"\NinjaCat"* mens resten av filene vil være katalogreferanser til de to andre image. Som vist i figur 24 får vi en lagstruktur, hvor webapplikasjonen i lag 2 er avhengig av web serveren i lag 1, som igjen er avhengig av Container OS Image i lag 0

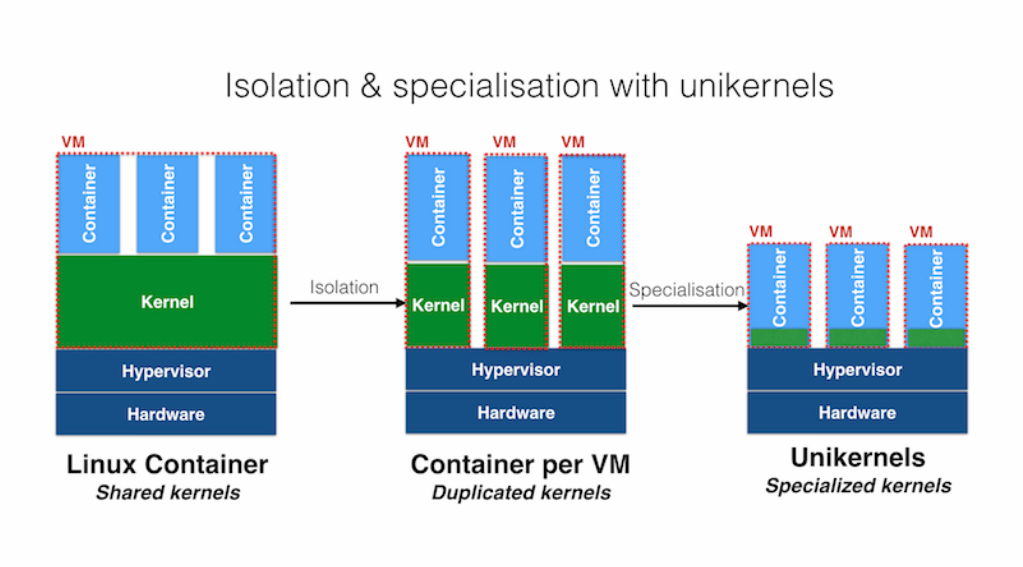
KAPITTEL

7

FRAMTIDEN FOR DOCKER OG CONTAINERTEKNOLOGI

7.1 Unikernel og Docker

Unikernel Systems ble i Januar en del av Docker [66]. Unikernel er et system som kjører en hypervisor rett på hardware, og kjører virtuelle maskiner på den som kun har de bibliotekene og runtimene man trenger for å kjøre en applikasjon [67]. Dermed reduseres overheaden sammenlignet med å kjøre et fullstendig operativsystem. Docker ønsker at unikenels skal optimaliseres til å fungere med Docker tools, slik at man kan få en enda større ressursbesparing og isolasjon mellom containere. Slik Docker fungerer nå, kjøres det først opp en container med operativsystemet applikasjonene man kjører krever. Så hvis man skal kjøre en IIS server, opprettes det en container med Windows Server
Core, så opprettes det en container hvor IIS blir installert. Begge kjører samtidig, og kun endringene mellom dem lagres i IIS-containeren. Dersom brukeren ønsker å kjøre enda en applikasjon i Windows Server Core opprettes det da kun en ny container som fungerer på samme måte som den IIS er installert i.

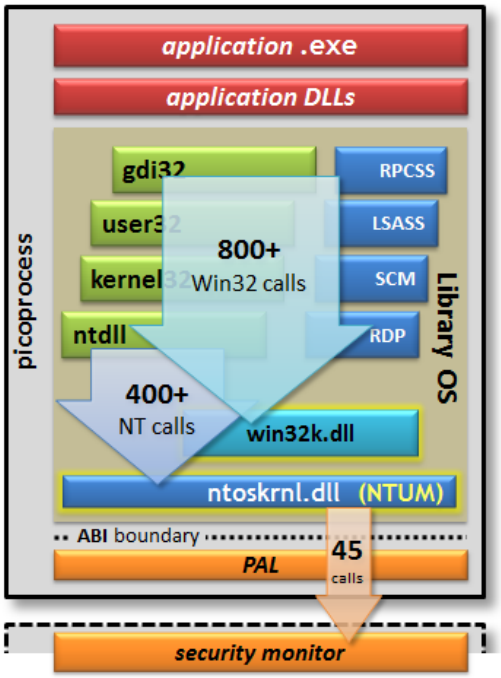


Figur 25: Forskjellen på containere og unikernels [68]

Figur 25 viser hvordan containere fungerer på Linux i dag, der flere containere deler på en kernel i en virtuell maskin. Deretter vises hvordan isolerte containere à la hyper-v containere på Windows Server 2016 fungerer, der hver container kjører i en virtuell maskin med sin egen kernel. Og til sist, hvordan unikernel-containere vil fungere, der hver container/applikasjon vil kjøre i en virtuell maskin med akkurat de delene av operativsystemet de trenger.

Med et unikernel system, vil kun de avhengighetene hver applikasjon har behov for kjøres. Dette vil redusere kompleksiteten og ressurstapet som inngår i et moderne operativsystem. En mailserver trenger for eksempel sjelden lyddriverne, eller støtten for IDE disker, som er inkludert i Windows eller Ubuntu. Fra et sikkerhetsmessig ståsted er det også en klar fordel, ettersom angrepsflaten vil bli sterkt redusert. Det er vanskelig å angripe sårbarheter i programvare som ikke er installert. Det finnes mange forskjellige

prosjekter som forsøker å lage slike systemer, som bygger på forskjellige språk og retter seg mot forskjellige virtualiseringsteknologier. Blant annet MirageOS, Clive og HalVM [69].



Figur 26: Microsofts illustrasjon for systemkall i en picoprocess [67]

Microsoft sitt svar på unikernels er Microsoft Drawbridge, som lager såkalte picoprocesses. Dette er blant annet teknologien som ligger til grunn for at Bash kommer til Windows 10 [70]. Figur 26 viser Microsofts design av en slik picoprocess. Der det er en application binary interface (ABI i illustrasjonen) med 45 forskjellige systemkall som blir mottatt av en security monitor, som har samme rolle som en hypervisor i figur 25.

KAPITTEL

8

YTELSESTESTING

Ytelsestesting, eller benchmarking, er en metode for å vurdere ytelsen til et gitt produkt. Dette blir gjort ved å kjøre ett eller flere programmer som kjører ulike typer tester på eksempelvis en prosessor. Det skilles mellom to typer ytelsestester: syntetisk og real-world.

Syntetisk

Dette er den vanligste formen for ytelsestesting. Denne typen testing utfører ingenting annet enn å produsere tall og data til å sammenligne med andre utførte tester. Når testen er utført får produktet en poengsum som man kan sammenligne med andre tester. Testen alene sier svært lite om ytelsen til produktet, men det er en god metode for å sammenligne hvordan ulike produkter klarer å utføre identiske oppgaver.

Real World

I motsetning til syntetisk ytelsestesting produserer som regel ikke real-world testing en målbar poengsum etter ferdig testing. I real-world testing brukes det ofte programmer som i utgangspunktet ikke er designet for ytelsestesting, for eksempel kontorapplikasjoner, spill og 3D-rendering. Real-world testing er vanskeligere å tolke da det er langt flere variabler som må vurderes i en slik type testing [71].

Målet med denne delen er å se på ytelsen til en Windows Container, ytelsestap på den fysiske maskinen ved drift av et containersystem under last og hvordan dette yter i forhold til virtuelle maskiner. Det finnes allerede lignende tester, men da på Linux-baserte systemer [3]. Det er derfor interessant å se hvordan denne ytelsen er i en Windows Container, og hvor mye driften av et slikt containersystem tar av ressursene til den fysiske maskinen, i forhold til virtuelle maskiner. Dette er spesielt interessant på grunn av at i gjeldende versjon av Windows Server 2016 er det nødvendig å ha en virtuell maskin som container host. Dette leder dog til en ekstra grad av usikkerhet, ettersom det ikke er en direkte sammenligning av teknologiene.

I alle de ulike testene blir det utført 30 tester, og tabellene viser et gjennomsnitt av disse testene. De svarte vertikale linjene, er standardavviket i disse 30 testene.

8.0.1 Verktøy

Her er en kort gjennomgang av verktøyene som har blitt brukt til testene i dette kapittelet.

Geekbench

Geekbench™ er et multi-platform prosessor-benchmarkverktøy som kan teste ytelsen i enkeltkjerne- eller multikjerneløsninger [72]. Den kjører en rekke ulike tester, blant annet AES, Twofish, SHA1, SHA2, JPEG Kompresjon, Dijkstra i tillegg til minnetester ved hjelp av funksjoner som Stream Scale og Stream Copy. Resultatene er uavhengig av hvilket operativsystem du kjører og støtter Windows, Linux, OSX, Android og iOS. Sluttresultatene blir gitt ut fra en baseline score på 2500 som representerer ytelsen til en Intel Core i5-2520M @ 2.50GHz.

Gruppen valgte å bruke Geekbench™ da det var behov for et ytelsestestingsverktøy som ikke krever GUI-funksjonalitet, noe som Windows Container ikke støtter. Da Geekbench™ kan opereres fra en kommandolinje og den utfører en grundig test, falt valget på dette.

En eksempelutskrift fra Geekbench™ finnes i vedlegg I. Rådataene fra testingen som

ligger til grunn for grafene videre i dette kapittelet finnes i vedlegg E.

Diskspd

Microsoft Diskspd er et verktøy for syntetisk testing av diskytelse [73]. Diskspd kjører fra kommandolinjen, noe som betyr at det enkelt kan scriptes til å utføre en mengde tester og at den kan kjøres i en container. Verktøyet produserer syntetiske arbeidsbelastninger med en rekke ulike parametere, blant annet I/O-størrelse, tilfeldig eller sekvensiell i tillegg til valget mellom kun lese, lese/skrive og kun skrive [74]. Microsoft anbefaler diskspd for testing av lagring til SQL server [75].

NTttcp

NTttcp er et verktøy for å måle nettverksytelse på Windows-baserte systemer [76]. Dette verktøyet ble valgt på grunn av behovet for et verktøy som kan kjøres fra kommandolinje. Microsoft har utviklet NTttcp og anbefaler dette til testing av Windows-baserte maskiner [77].

8.1 Testresultater

Generelt om Geekbench-testene

Geekbench™ 3 bruker en rekke ulike tester og arbeidsbelastninger for å måle ytelse, disse blir delt opp i tre ulike deler:

Integer performance

Integer arbeidsbelastning måler produktets evne til å kjøre intense prosessoraktiviteter som bruker en stor mengde integerinstruksjoner [78]. All programvare bruker en stor mengde integerinstruksjoner, og et godt resultat her vil derfor indikere god ytelse totalt. Belastninger som blir kjørt er blant annet kryptering i form av AES, Twofish og SHA1 [79].

Floating point performance

Floating point arbeidsbelastninger utfører en rekke intense prosessoraktiviteter der tung bruk av floating point operasjoner er vektlagt [78]. Nesten all programvare bruker "floating point"-operasjoner til en viss grad, men det er brukes spesielt ved opprettes av digitalt materiale, spill og andre databehandlingsprogrammer med en høy ytelse. Belastninger som blir kjørt er blant annet Black-Scholes, Mandelbrot, N-Body og Ray trace [79].

Memory performance

Arbeidsbelastningene som blir kjørt her måler båndbredden på minnet. Programvare som arbeider med store mengder data avhenger av å ha en god minnebåndbredde for å holde prosessoren opptatt. Arbeidsbelastningene er blant annet STREAM copy som måler hvor raskt produktet kan kopiere store mengder data i minne [79] [80].

Til slutt blir det produsert en Geekbench™ total poengsum. Dette er et vektet gjennomsnitt av de tre typene arbeidsbelastning og gir en heltallsverdi som kan måles mot andre produkter.

Generelt om Diskspd-testene

På I/O-testingen med Diskspd skilles det mellom to typer tester - Tilfeldig og sekvensiell:

- Tilfeldig: Lesing eller skriving til tilfeldige steder i filen for hver operasjon.
- Sekvensiell: Lesing eller skriving til disk hvor den starter på begynnelsen av filen og etterfølgende operasjoner ligger sekvensielt posisjonert etter den første.

På begge I/O -testene måles de samme verdiene:

- MB/s: Måler gjennomstrømming(throughput), altså antall I/O per sekund ganget med I/O-størrelse
- IOPS: "Input/Output per second". Antall operasjoner per sekund.
- Latency: Tiden det tar fra et I/O-kall blir sendt til operasjonen er fullført.

Både VM og container kjører med 12GB minne og tilgang til alle kjerner i CPU.

Det blir brukt en alternativ fysisk maskin til I/O-testen, ettersom maskinen lånt av Ikomm kun har en disk. For å minimere feilmarginer som følge av kjøringen av maskinen, er det ønskelig å kjøre disk-tester på en disk som ikke kjører operativsystemet. Spesifikasjonene til denne alternative maskinen finnes i tabell 2. Rådata fra testene finnes i vedlegg G, mens en eksempelkjøring fra diskspd finnes i vedlegg J.

Prosessor	Intel(R) Core(TM) i7-3610QM CPU @ 2.30GHz
Minne	16GB 1600MHz
Harddisk 1	160GB 5400RPM (Kjører operativsystemet)
Harddisk 2	500GB 5400RPM (Brukt til testing)

Tabell 2: Spesifikasjoner på alternativ maskin

Generelt om NTttcp-testen

I nettverkstesten ønsket gruppen å finne ut hvorvidt det er tap av hastighet på en container satt opp i mot VM og eventuelt hvor stort tap. Av de fire støttede nettverksmodusene til Windows Container benyttes network address translation mode (delkapittel 6.3.3). Nettverkslinken mellom maskinene er 1GigE fiber og målingene vil være av gjennomstrømming fra serveren til en annen maskin innenfor nettverket på NTNU i Gjøvik, målt i mbps. Det er viktig at mottakermaskin har minst like mange tråder som sender. Sender har åtte fysiske kjerner og mottakermaskin har bare seks kjerner, men med hyper-threading (tabell 3). Grunnet brannmur var det kun mulig å teste med serveren som sender. Et eksempel fra en kjøring av NTttcp finnes i vedlegg K, og rådata fra testene finnes i vedlegg F.

Hovedkort	X99-UD-CF med 1GbE Intel Chipset
Prosessor	Intel Core i7 5820K 3.30 GHz
Minne	16GB DDR4 2133MHz

Tabell 3: Spesifikasjoner på mottakermaskin

8.1.1 Usikkerheter

Et stort usikkerhetsmoment ved testingen er at container host er en virtuell maskin. Dette medfører at det ikke blir like forhold mellom VM og container, da container er kjørt i virtuell maskin mens VM er kjørt direkte på host. Da container host blir en virtuell maskin fører dette til ekstra overhead. Med andre ord er det ikke mulig med en "apples

to apples" sammenligning mellom container og VM.

Selv om det ble gjort tiltak for å begrense mengden prosesser som kjørte under ytelsestestingen kan andre prosesser ha påvirket testresultatet, ettersom testene ble gjort på et kjørende system.

Host kjørte Windows Server 2016 - Technical Preview 4. Denne kan skape noe usikkerhet da det er en testversjon og ikke et komplett produkt.

Tidtakingen av de virtuelle maskinene ble gjort med stoppeklokken på en telefon, så den er ikke like nøyaktig som tidtakingen av containerne. Tidene er dog forskjellige nok til at det er klart hvilken verdi som er størst selv med denne usikkerheten.

I testingen av nettverksytelse kan resultatene bli påvirket av annen nettverkstrafikk, enten på sender/mottakermaskin, eller generelt på nettverket. Dette ble gjort fra serveren på campus til studentboligene på Kallerud. Det er fiber mellom disse to punktene, men det er fortsatt en mulighet for forstyrrelser.

I testingen av både nettverk og I/O, blir det brukt software som ikke er verifisert at fungerer optimalt på Windows Server 2016, og heller ikke på Windows 10. Dette er viktig, ettersom Windows 10 er grunnlaget for arkitekturen i Windows Server 2016. Det kan være uforutsette feil i kjøringen av NTttcp og diskspd på Windows Server 2016 som eventuelt påvirker resultatet.

8.1.2 Oppstarts- og stop-tider

For å ta tiden på containerne ble det brukt *Measure-Command* cmdleten i Powershell som måler tiden det tar for en kommando å kjøre. Nedenfor er den nøyaktige kommandoen som ble kjørt. (Med *Stop-Container* og *Start-Container* byttet om for å måle start-tid)

```
for ($i=1; $i -le 30; $i++) {$a = Measure-Command
{stop-container test1};        Add-Content .\testresultat $a;
start-container test1}
```

Vi definerte oppstart på virtuelle maskiner som tiden fra man trykker på start-knappen til beskjeden: "press ctrl-alt-del to log in" kommer opp på skjermen, basert på en artikkel av Scott Lowe [81]. Målingen av oppstartstid på de virtuelle maskinene og ble gjort med stoppeklokkefunksjonen på en telefon, så er ikke like nøyaktig som tidtakingen av containerne. Resultatene som ligger til grunn for denne tabellen finnes i vedlegg H

		Tider	
		Start	Stop
VM	Tider	17,47s	2,58s
	Avvik	0,78s	0,12s
Container	Tider	3,5s	1,75s
	Avvik	0,14s	0,25s

Tabell 4: Start og stop tider for en virtuell maskin i Hyper-V og containere. Verdiene er gjennomsnittet etter 30 gjennomførte tester. Avvik er standardavviket gjennom disse 30 testene

Som tabell 4 viser, er containere mye raskere til å starte opp, og prosentvis relativt mye raskere til å slå seg av.

8.1.3 Testresultater fra Geekbench

Ytelsestest uten last

I første del av testingen ble det kjørt tester uten annen belastning enn den kjørende ytelsestesten.

Host

Dette er den fysiske serveren som ikke kjører noe annet enn Geekbench™. Operativsystem er Windows Server 2016 - Technical Preview 4.

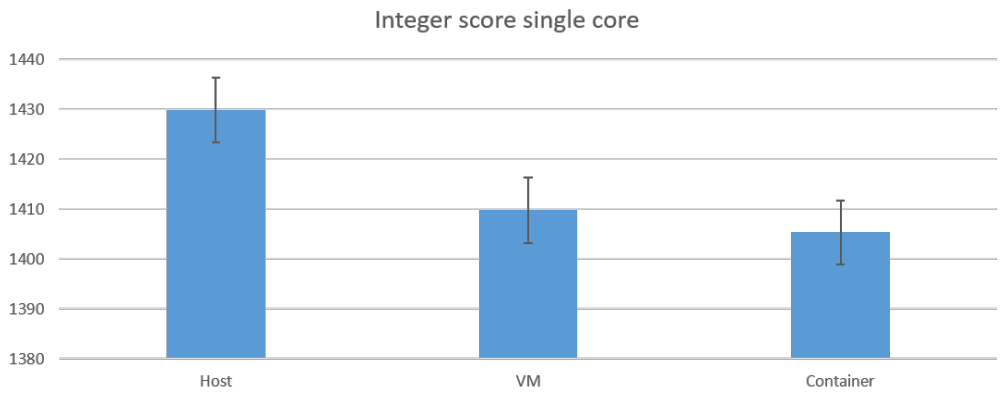
VM

En virtuell maskin kjørt i Hyper-V som kun kjører Geekbench™. Operativsystem er Windows Server Core.

Container

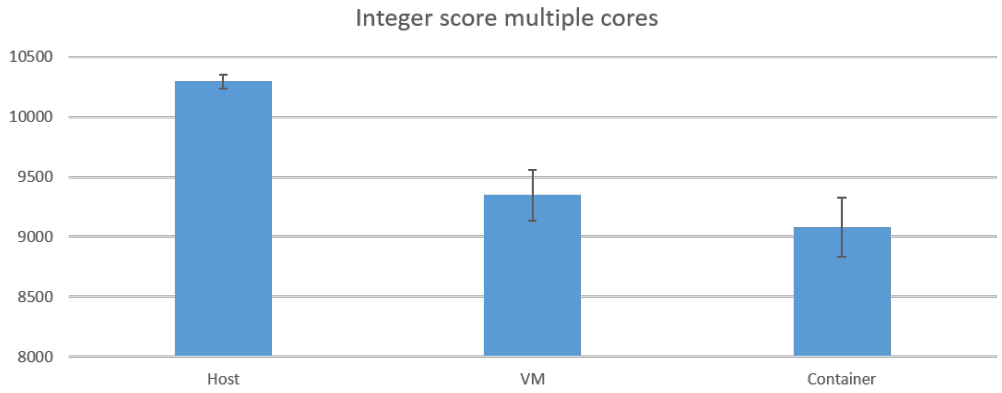
En container host som kjører en container som igjen kjører Geekbench™. Operativsystem er Windows Server Core.

Både VM og Container har tilgang til 12GB RAM og alle de åtte kjernene til CPU-en.



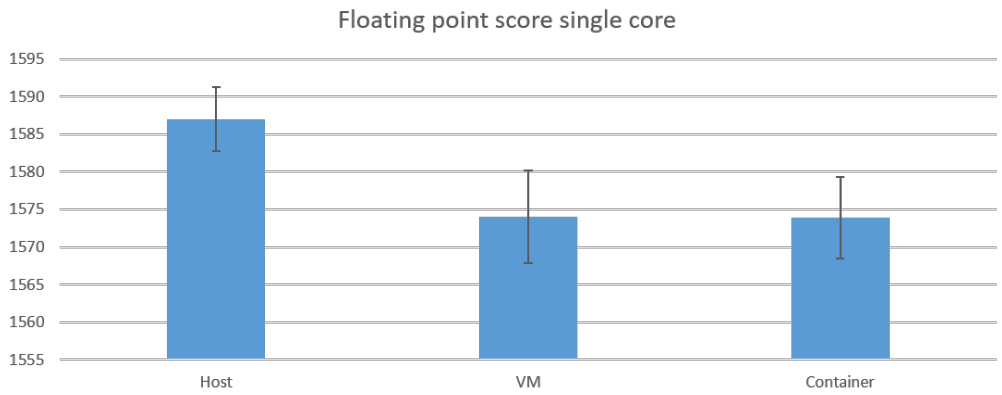
Figur 27: Poengsum av integeroperasjoner med en kjerne. Dette er gjennomsnittet etter 30 gjennomførte tester. Feilmarginslinjene indikerer standardavviket av disse testene.

Figur 27 viser ytelsen av de ulike enhetene på integeroperasjoner på en enkelt kjerne. Host er logisk nok den som yter best med VM som nummer to i rekken. Nederst ligger Windows Container. Dette er ikke overraskende med tanke på at container hosten i dette tilfellet er det eneste som produserer last på maskinen. En kan se et klart skille mellom host og VM/container, og det er lett å se at det krever en mengde ressurser å skjule systeminformasjon fra hosten i tillegg til å virtualisere systemkomponenter. Standardavviket er tilnærmet likt på alle tre enhetene.



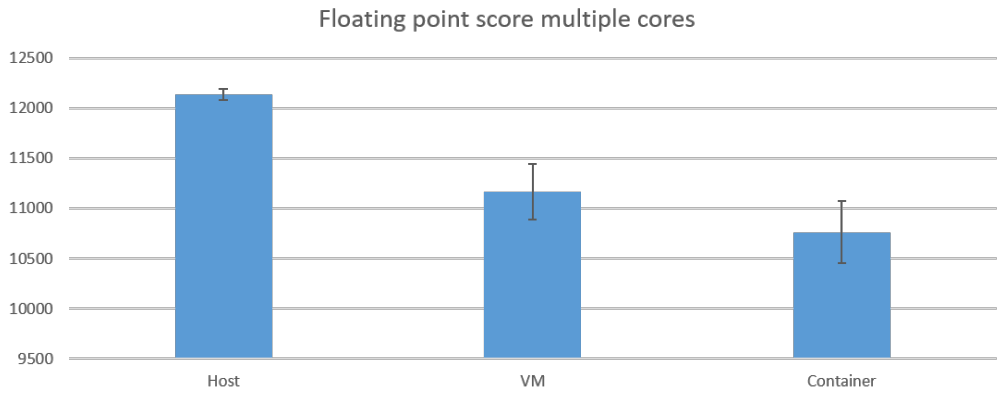
Figur 28: Poengsum av integeroperasjoner med åtte kjerner. Dette er gjennomsnittet etter 30 gjennomførte tester. Feilmarginslinjene indikerer standardavviket av disse testene.

Figur 28 viser ytelsen av de tre ulike enhetene på integeroperasjoner på åtte kjerner. Forholdet mellom de ulike enhetene er tilnærmet lik som på en enkelt kjerne med container som svakeste enhet og host som beste. Grunnen til at standardavviket er høyere på VM og container er trolig den ekstra overheaden som produseres når enhetene først sender kall til sine virtuelle kjerner for at hypervisor deretter skal sende instruksjonene til de ulike kjernene.



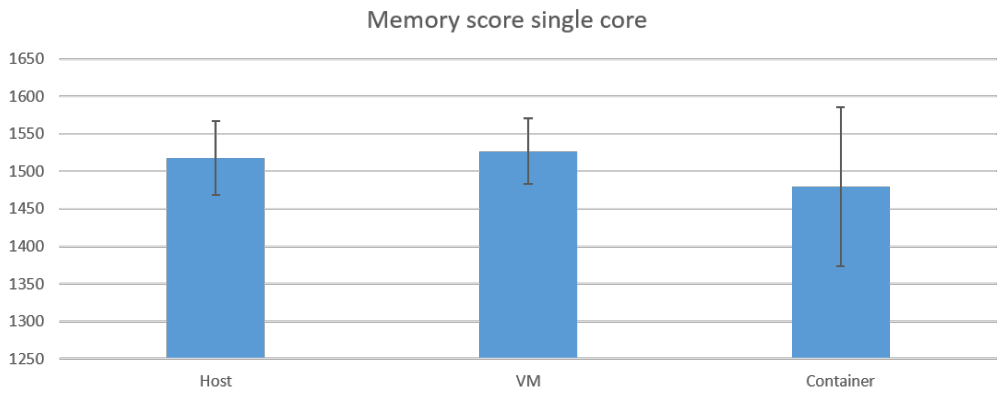
Figur 29: Poengsum av “floating point”-operasjoner med en kjerne. Dette er gjennomsnittet etter 30 gjennomførte tester. Feilmarginslinjene indikerer standardavviket av disse testene.

Figur 29 viser ytelsen av de ulike enhetene med “floating point”-operasjoner med en enkelt kjerne. VM og container er tilnærmet lik på ytelse, men VM har et noe høyere standardavvik.



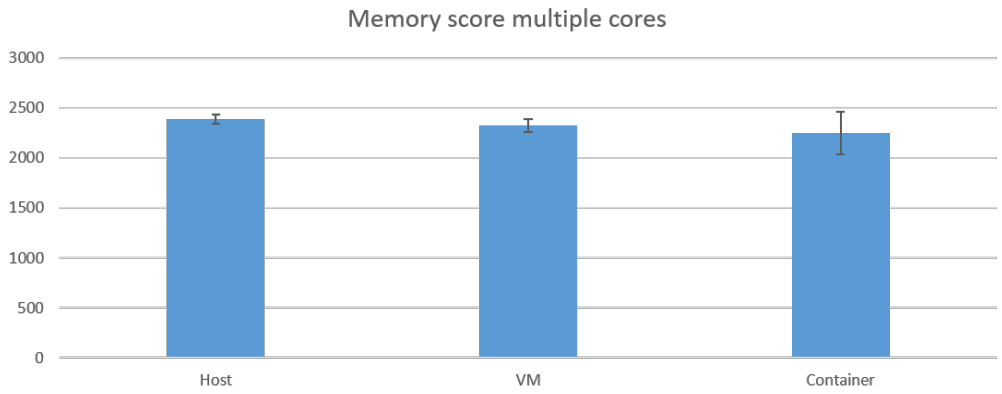
Figur 30: Poengsum av “floating point”-operasjoner med åtte kjerner. Dette er gjennomsnittet etter 30 gjennomførte tester. Feilmarginslinjene indikerer standardavviket av disse testene.

I figur 30 vises ytelsen av de ulike enhetene med floating point-operasjoner på åtte kjerner. Host har nok en gang det beste resultatet og laveste standardavviket. Container kommer her dårligst ut med lavest poengsum og høyest standardavvik.



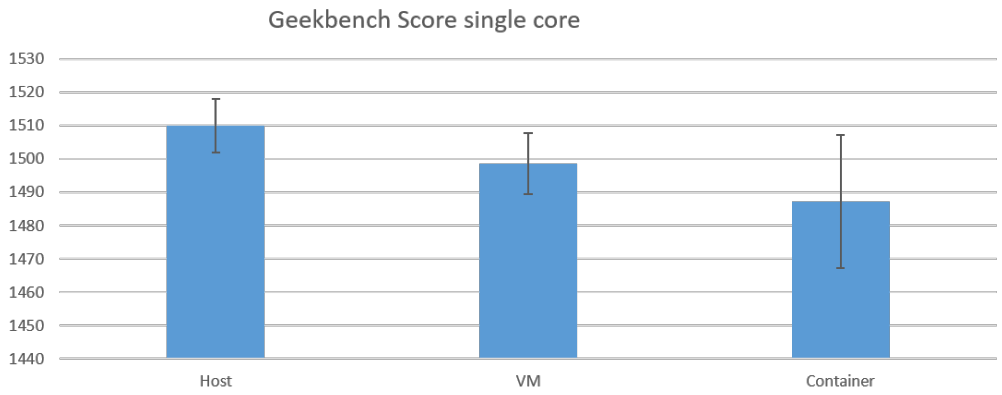
Figur 31: Poengsum av minneytelse med en kjerne. Dette er gjennomsnittet etter 30 gjennomførte tester. Feilmarginslinjene indikerer standardavviket av disse testene.

I figur 31 vises ytelsen på minnebåndbredde på de ulike enhetene. Det er den første av testene hvor VM og host er tilnærmet lik i ytelse og med et lignende standardavvik. Containeren scoret dårligst med et særdeles stort standardavvik. Under et par av testene på containeren fikk den særdeles dårlige resultater. Hvorfor dette skjedde er uvisst.



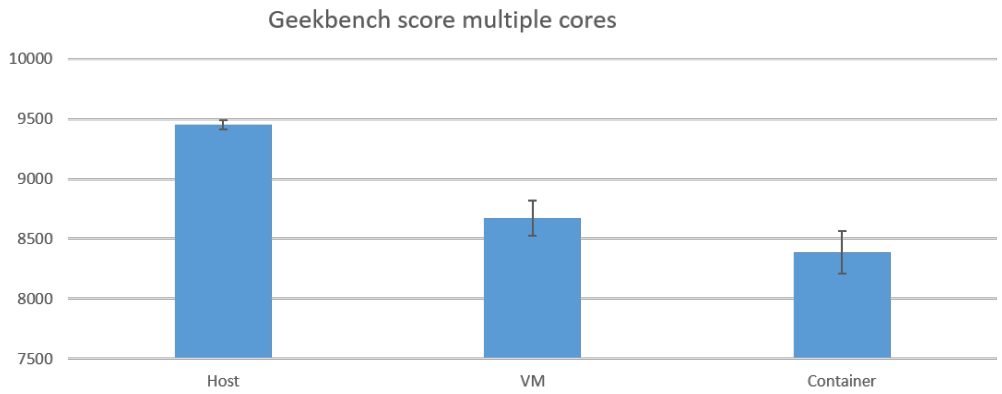
Figur 32: Poengsum av minneytelse med åtte kjerner. Dette er gjennomsnittet etter 30 gjennomførte tester. Feilmarginslinjene indikerer standardavviket av disse testene.

I figur 32 vises minneytelse med åtte kjerner på de ulike enhetene. Her er ikke skillet mellom de ulike enhetene av samme størrelse som tidligere, men også her har container det største standardavviket.



Figur 33: Total poengsum av alle de ulike testene på en kjerne. Dette er gjennomsnittet etter 30 gjennomførte tester. Feilmarginslinjene indikerer standardavviket av disse testene.

I figur 33 vises den totale poengsummen for de ulike enhetene når det testes på en enkelt kjerne. Host er logisk nok vinneren med høyeste totale poengsum. Container er den svakeste enheten og den med størst standardavvik. Dette er heller ikke overraskende. Container er som kjent den eneste av enhetene som skaper noe last, med tanke på at container host kjører en container. Dette bidrar til flere lag som må gjennomgås og mer overhead, noe som delvis forklarer det store standardavviket.



Figur 34: Total poengsum av alle de ulike testene på åtte kjerner. Dette er gjennomsnittet etter 30 gjennomførte tester. Feilmarginslinjene indikerer standardavviket av disse testene.

I figur 34 vises den totale poengsummen for de ulike enhetene når det testes på åtte kjerner. Container er også her den svakeste enheten, i enda større grad enn på en enkelt kjerne. Som nevnt tidligere har host minst standardavvik på grunn av lite overhead og container/VM mer på grunn av flere lag å gå gjennom og derfor mer overhead.

Ytelsestest under last

For å få belastning på CPU-en, ble det kjørt et script som listet ut 100 filer kontinuerlig ved hjelp av *Get-ChildItem* i de ulike enhetene. Geekbench™ ble kjørt på host i alle

tilfeller for å måle hvor mye ressurser som går med til å kjøre de ulike enhetene med last.

Scriptet som ble kjørt:

```
while(1){Get-ChildItem -Filter "file*.txt"}
```

Host

Fem tråder som kjører script, I tillegg til èn versjon av Geekbench™. Operativsystem er Windows Server 2016 - Technical Preview 4.

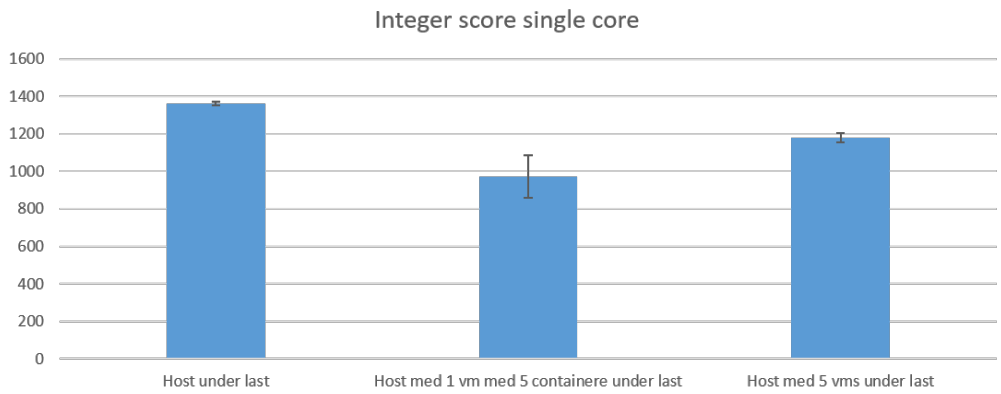
VM

Fem virtuelle maskiner som kjører script, Geekbench™ på host. Operativsystem for de virtuelle maskinene er Windows Server Core.

Container

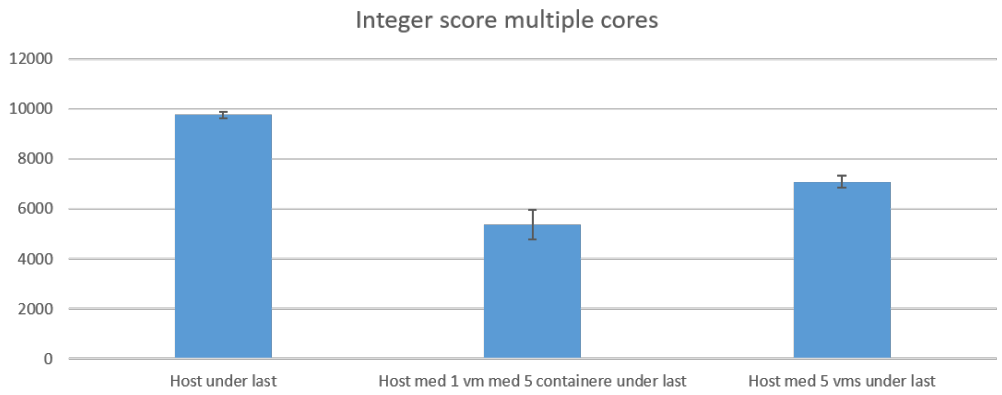
Containerhost som kjører fem containere. Hver container kjører script. Geekbench™ på host. Operativsystem for containere og container host er Windows Server Core.

Hyper-V manageren rapporterte stabil last rundt 66% på container hosten og rundt 7% på hver av de virtuelle maskinene, mens task manager rapporterte 55% cpu belastning på hosten.



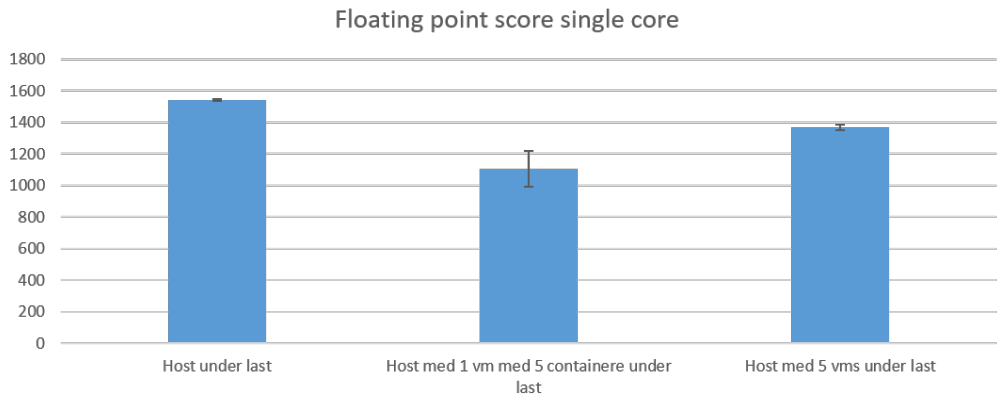
Figur 35: Poengsum av integeroperasjoner med en kjerne. Dette er gjennomsnittet etter 30 gjennomførte tester. Feilmarginslinjene indikerer standardavviket av disse testene.

I figur 35 vises ytelsen av integeroperasjoner på de ulike enhetene med èn kjerne. Det er ikke overraskende at host gir det beste resultatet, men at VM gjør det bedre enn containere er uforventet. Det er et lavt standardavvik på både VM og host, men det er betraktelig høyere på container. En av årsakene til dette er trolig den ekstra overheaden som produseres ved at container kjøres i en virtuell maskin. En annen medvirkende årsak kan være at container host rapporterer høyeste stabile last.



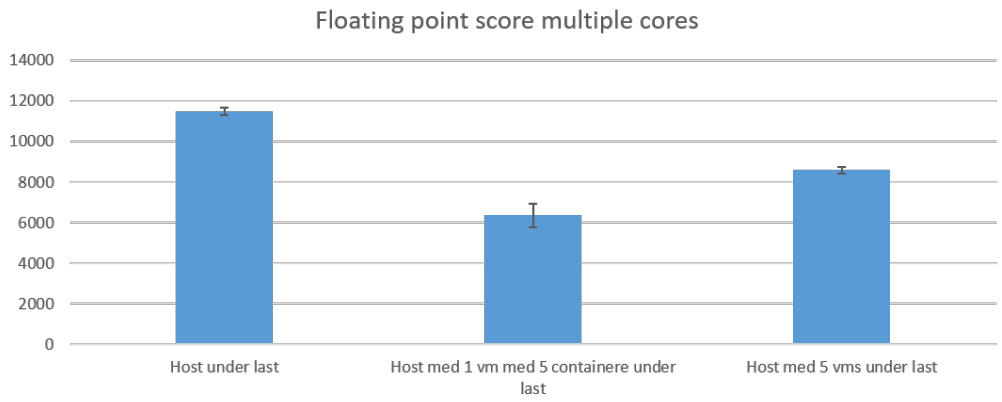
Figur 36: Poengsum av integeroperasjoner med åtte kjerner. Dette er gjennomsnittet etter 30 gjennomførte tester. Feilmarginslinjene indikerer standardavviket av disse testene.

Som figur 36 viser er det klart at VM og container ikke klarer å utnytte de åtte kjernene like godt som host og får en lavere poengsum her sett i forhold til ytelsen på èn kjerne. Forholdet mellom container og VM er relativt like på èn- og flerkjernet ytelsestest med integeroperasjoner.



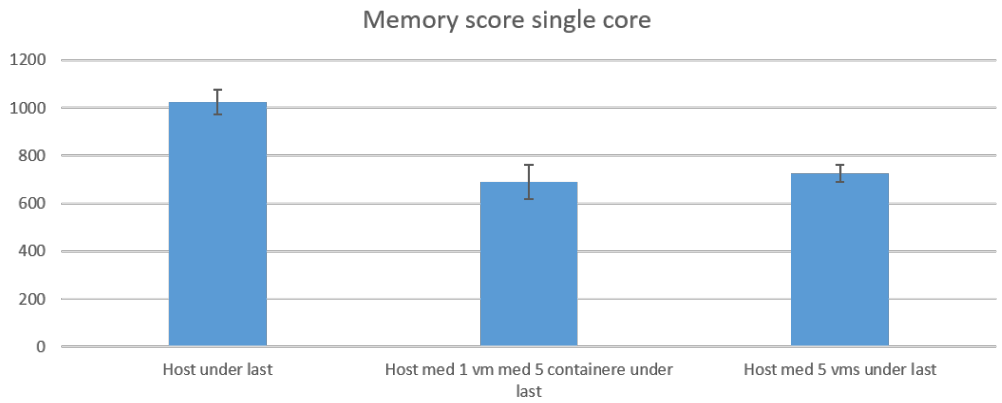
Figur 37: Poengsum av “floating point”-operasjoner med en kjerne. Dette er gjennomsnittet etter 30 gjennomførte tester. Feilmarginslinjene indikerer standardavviket av disse testene.

I ytelsestest av “floating point”-operasjoner med èn kjerne er resultatene relativt like som i èn-kjernet testing av integeroperasjoner, som vist i figur 37 . Nok en gang kommer container svakest ut i testen med det høyeste standardavviket.



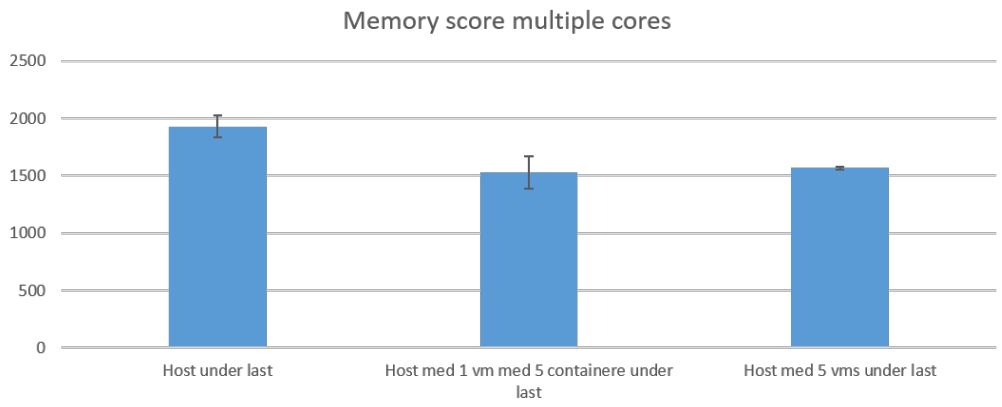
Figur 38: Poengsum av “floating point”-operasjoner med åtte kjerner. Dette er gjennomsnittet etter 30 gjennomførte tester. Feilmarginslinjene indikerer standardavviket av disse testene.

I denne testen av “floating point”-operasjoner med åtte kjerner yter container tilnærmet halvparten så bra som host som illustrert i figur 38 . VM sitt forhold til host er litt svakere enn på integeroperasjoner på åtte kjerner.



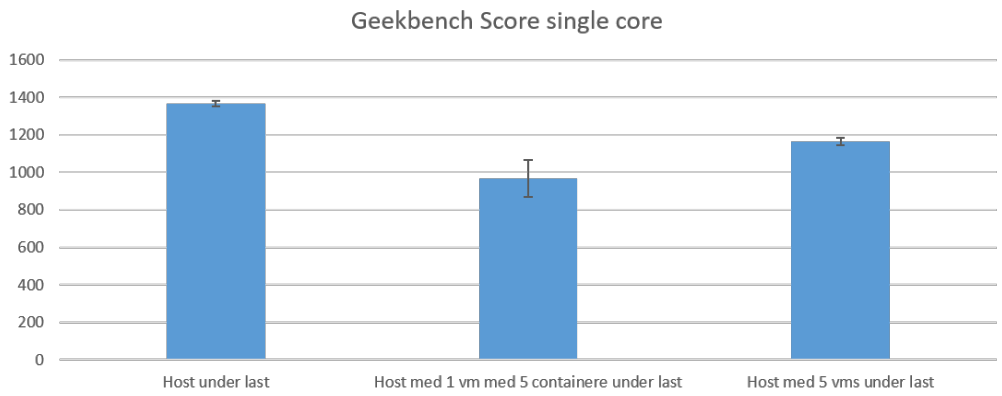
Figur 39: Poengsum av minneytelse med en kjerne. Dette er gjennomsnittet etter 30 gjennomførte tester. Feilmarginslinjene indikerer standardavviket av disse testene.

Som vist i figur 39 er ytelsen til containere mer positiv i forhold til VM når det kommer til minnebåndbredde med èn kjerne. VM og container er nesten helt like på ytelse, men container har et større standardavvik.



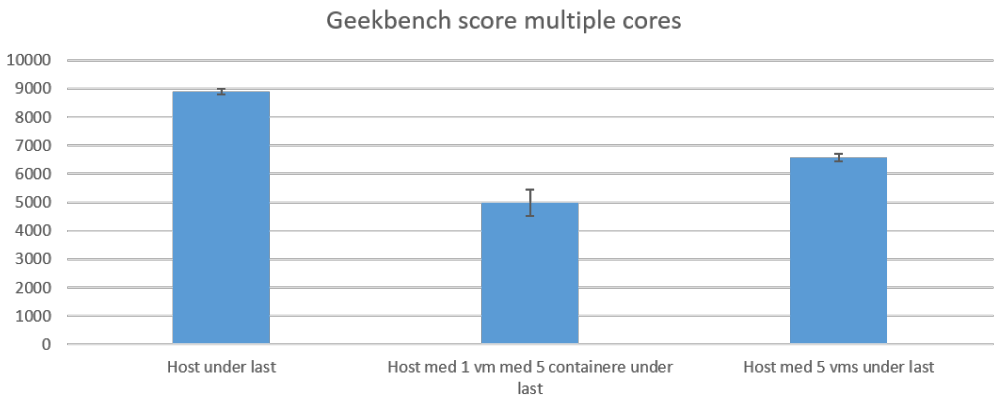
Figur 40: Poengsum av minneytelse med åtte kjerner. Dette er gjennomsnittet etter 30 gjennomførte tester. Feilmarginslinjene indikerer standardavviket av disse testene.

På samme måte som i testen av minnebåndbredde med èn kjerne yter VM og container tilnærmet likt. Dette blir vist i figur 40 . Det er interessant å legge merke til at VM og container er nærmere host i ytelse på åtte kjerner enn på èn kjerne.



Figur 41: Total poengsum av alle de ulike testene på en kjerne. Dette er gjennomsnittet etter 30 gjennomførte tester. Feilmarginslinjene indikerer standardavviket av disse testene.

Total poengsum for ytelsestester kjørt på èn kjerne vises i figur 41. Forholdet mellom VM og host er tilnærmet det samme som mellom VM og container. Container har det desidert største standardavviket og drar mest ressurser fra host ved å kjøre scriptene.



Figur 42: Total poengsum av alle de ulike testene på åtte kjerner. Dette er gjennomsnittet etter 30 gjennomførte tester. Feilmarginslinjene indikerer standardavviket av disse testene.

Den totale poengsummen på åtte kjerner viser tydelig at VM og container yter dårligere i forhold til host, som vist i figur 42 . Her ender også containere som taperen med å være den mest ressurskrevende og med høyest standardavvik.

8.1.4 Disk

I alle grafene i dette delkapittelet er Host, VM og Container definert på følgende måte:

Host

Dette er en laptop som kun kjører diskspd (spesifikasjoner i 2). Operativsystem er Windows Server 2016 - Technical Preview 4 with Desktop Experience.

VM

En virtuell maskin med 12GB ram, og tilgang til 4 kjerner og 8 tråder, som kun kjører diskspd. Operativsystem er Windows Server Core.

Container

En virtuell container host med 12GB ram, og tilgang til 4 kjerner og 8 tråder, som kjører en container som igjen kjører diskspd. Operativsystem er Windows Server Core.

I/O-test med tilfeldig dataaksessering

I denne delen ønsker gruppen å undersøke ytelsen til container mot VM på tilfeldig I/O-håndtering . Testen blir kjørt på en fil på 5GB(-c5G) i 120 sekunder(-d120) for hver av de 30 testene. Det blir brukt 1 tråd (-t1)og tillatt antall operasjoner liggende i kø er 16 (-o16). Blockstørrelse er 64KB(-b64k) og caching(-h) er skrudd av. Det er i tillegg skrudd på logging av latency(-L), og en skrive-entropiverdi på 1GB(-Z1G) for skrive-delen. Skrivedelen og lesedelen blir gjort separat i tillegg til en test med 50/50 fordeling av lesing og skriving.

Kommandoen som blir kjørt for testen av skrivehastighet:

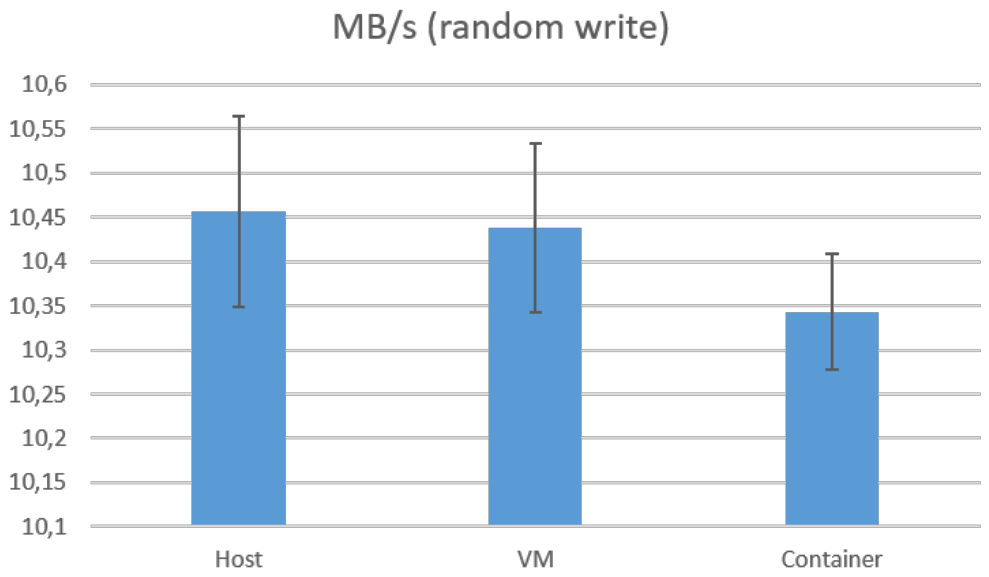
```
diskspd.exe -c5G -d120 -r -w100 -t1 -o16 -b64k -h -L -Z1G G:\TestFile1.dat
```

Kommandoen som blir kjørt for testen av lesehastighet:

```
diskspd.exe -c5G -d120 -r -w0 -t1 -o16 -b64k -h -L G:\TestFile1.dat
```

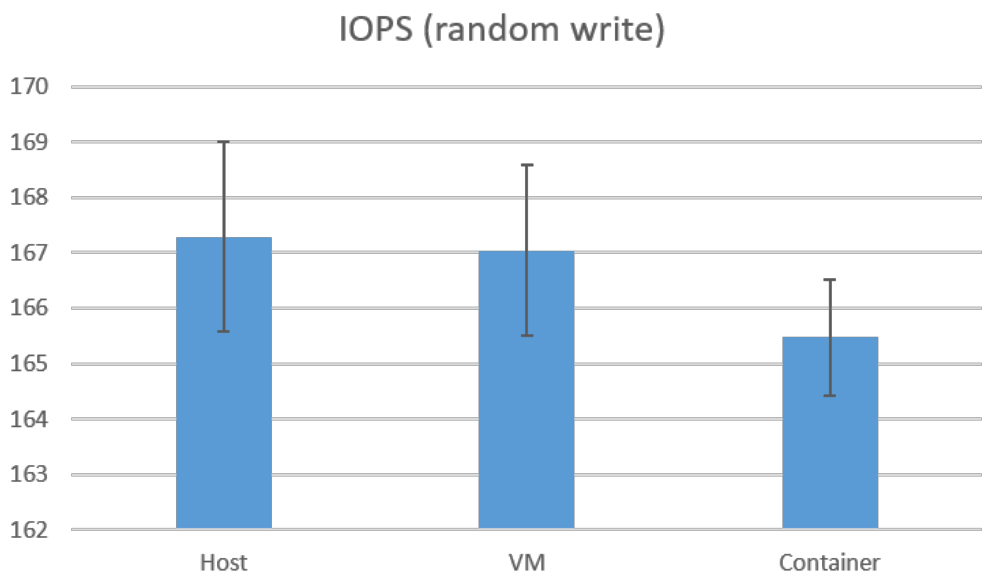
Kommandoen som blir kjørt for testen av blandet I/O:

```
diskspd.exe -c5G -d120 -r -w50 -t1 -o16 -b64k -h -L -Z1G G:\TestFile1.dat
```



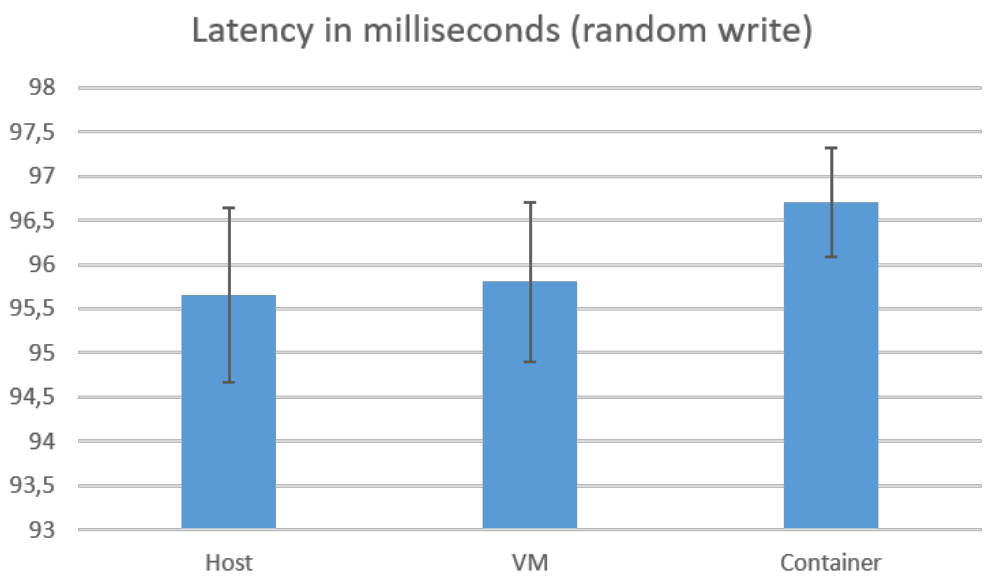
Figur 43: Gjennomstrømming på skriving tilfeldige steder i filen, målt i MB/s.

Som figur 43 viser ligger både host og VM høyt oppe med henholdsvis ca. 10,46 MB/s og 10,44 MB/s. Containeren henger litt etter med 10,34 MB/s. Container er den mest stabile med et standardavvik på 0,07.



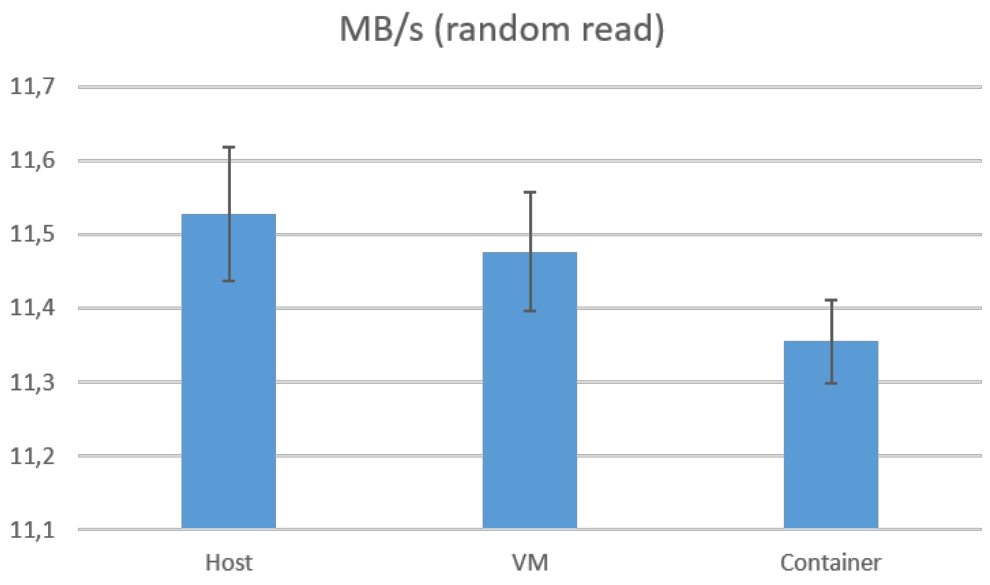
Figur 44: IOPS på skriving tilfeldige steder i filen.

På IOPS ser grafene (figur 44) tilnærmet lik ut i forhold til på gjennomstrømming. Host ligger på topp med 167,3 IOPS, VM med 167 IOPS og container med 165,5 IOPS.



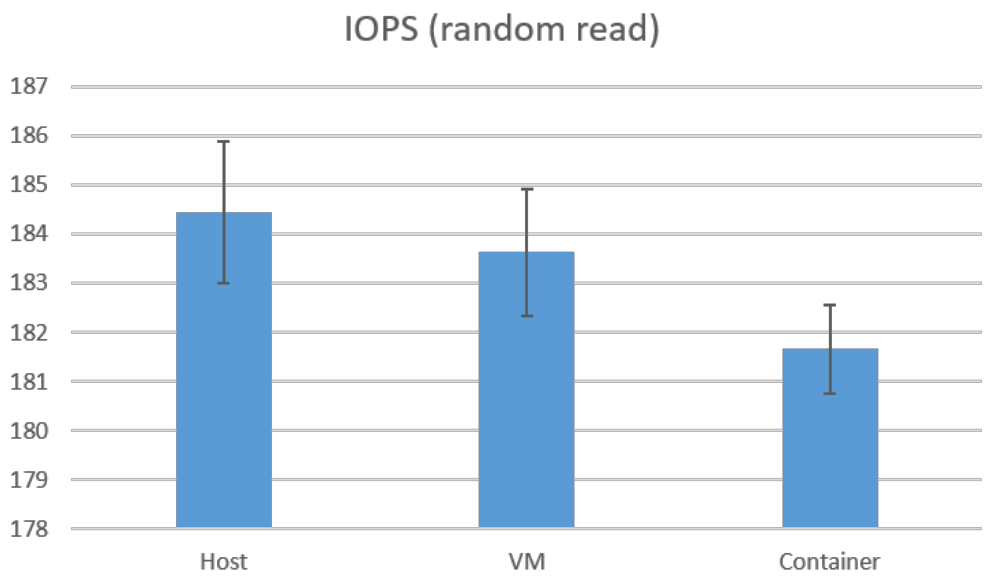
Figur 45: Latency på skriving tilfeldige steder i filen, målt i millisekunder.

Når det kommer til latency er grafene fortsatt veldig like sett i forhold til gjennomstrømming, bare motsatt (figur 45). Host har lavest latency med 95,65ms, VM i midten med 95,8ms og container med høyest latency, 96,7ms.



Figur 46: Gjennomstrømming på lesing tilfeldige steder i filen, målt i MB/s.

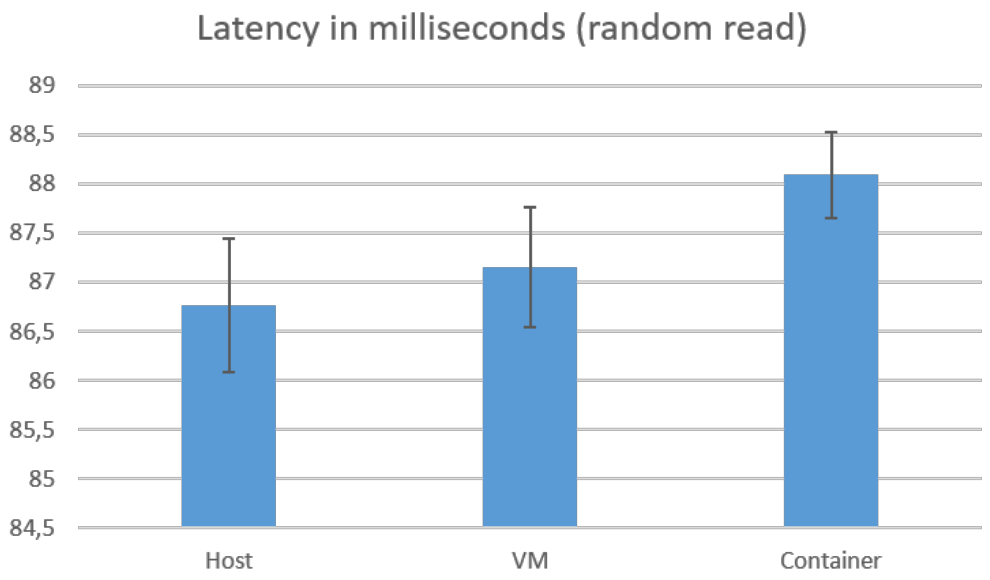
Figur 46 viser at host har en gjennomstrømming på 11,52 MB/s mens VM og container har henholdsvis 11,47 MB/s og 11,35 MB/s. Her også er container den mest stabile med et standardavvik på 0,06.



Figur 47: IOPS på lesing tilfeldige steder i filen.

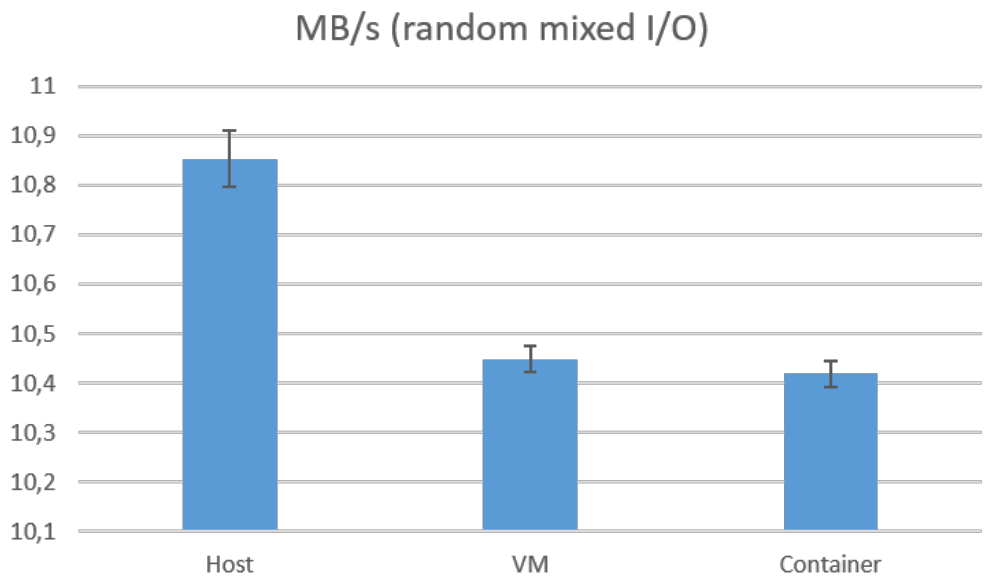
Forholdet mellom host, VM og container tilnærmet likt (figur 47) med gjennomstrøm på lesing. Host er best med 184,5 IOPS, VM i midten med 183,6 IOPS og container er

den dårligste med 181,7 IOPS.



Figur 48: Latency på lesing tilfeldige steder i filen, målt i millisekunder.

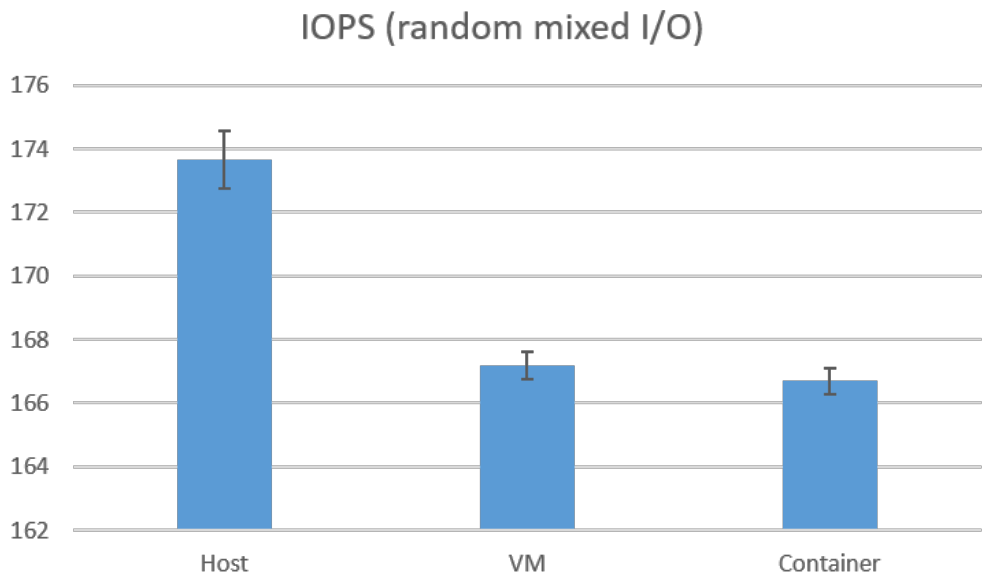
På samme måte som i skrivedelen har host laveste latency på 86,76ms, VM i midten med 87,15ms og container med høyeste latency, 88,08ms (figur 48).



Figur 49: Gjennomstrømming på lesing og skriving tilfeldige steder i filen, målt i MB/s.

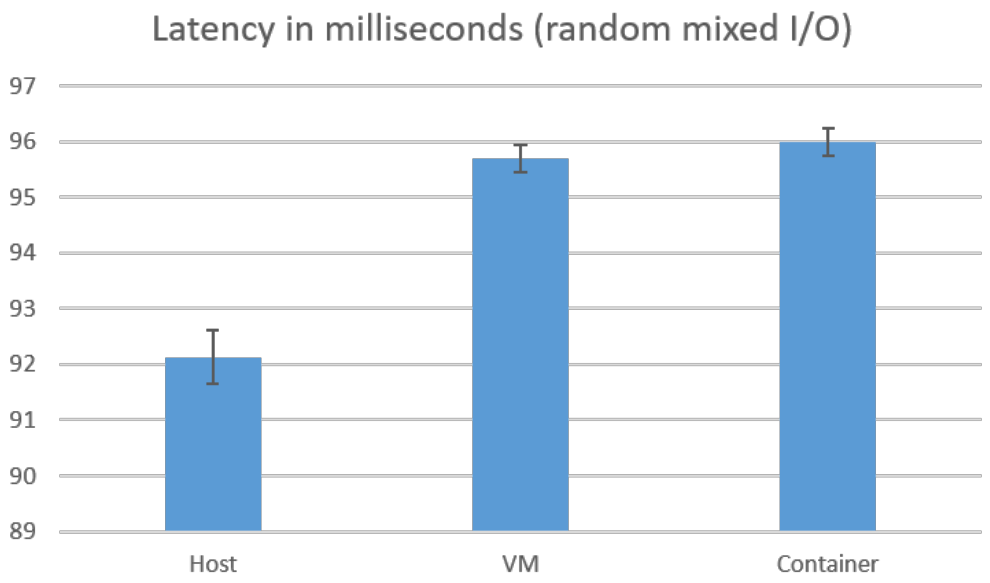
På gjennomstrømming når det kjøres les/skriv er VM og container tilnærmet like i

ytelse og forskjellene er små. Host, VM og container har henholdsvis 10,85 MB/s, 10,45 MB/s og 10,42 MB/s som vist i figur 49.



Figur 50: IOPS på lesing og skriving tilfeldige steder i filen.

Som vist i figur 50 er VM og container veldig lik på IOPS med ca 0,5 IOPS i forskjell. Host, VM og container har henholdsvis 173,66 IOPS, 167,18 IOPS og 166,70 IOPS.



Figur 51: Latency på lesing og skriving tilfeldige steder i filen, målt i millisekunder.

VM og container er på samme måte som med IOPS svært like med ca. 0,29ms i forskjell. Host, VM og container ligger henholdsvis på 92,13ms, 95,70ms og 95,99ms (figur 51).

I/O-test med sekvensiell dataaksessering

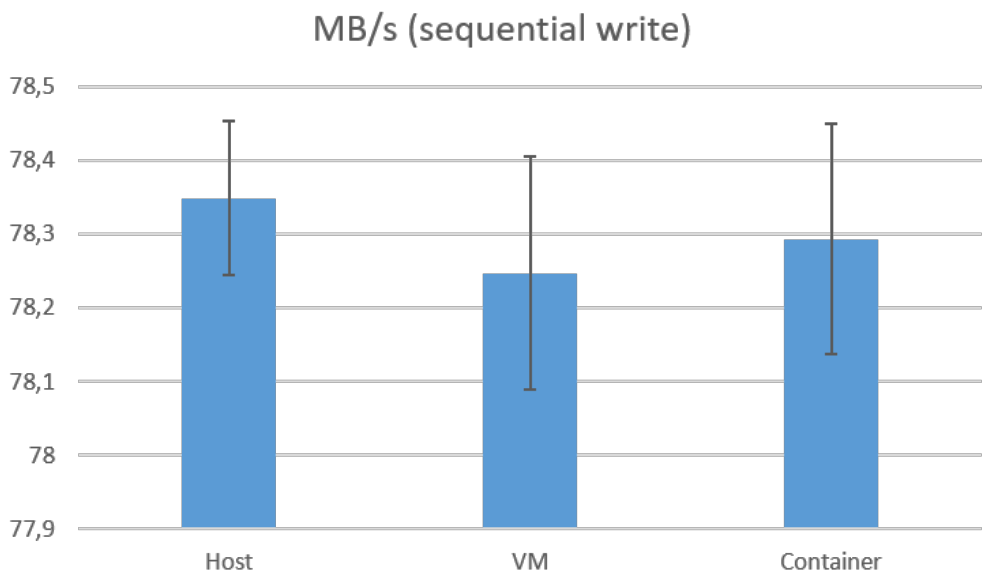
I denne delen ønsker gruppen å undersøke ytelsen til container mot VM på sekvensiell I/O-håndtering. Det blir brukt samme argumenter som i den tilfeldige I/O-håndteringen med unntak av at denne kjøres sekvensielt (-s). Skrivedelen og lesedelen blir gjort separat.

Kommando for skrivedelen:

```
diskspd.exe -c5G -d120 -s -w100 -t1 -o16 -b64k -h -L -Z1G g:\TestFile1.dat
```

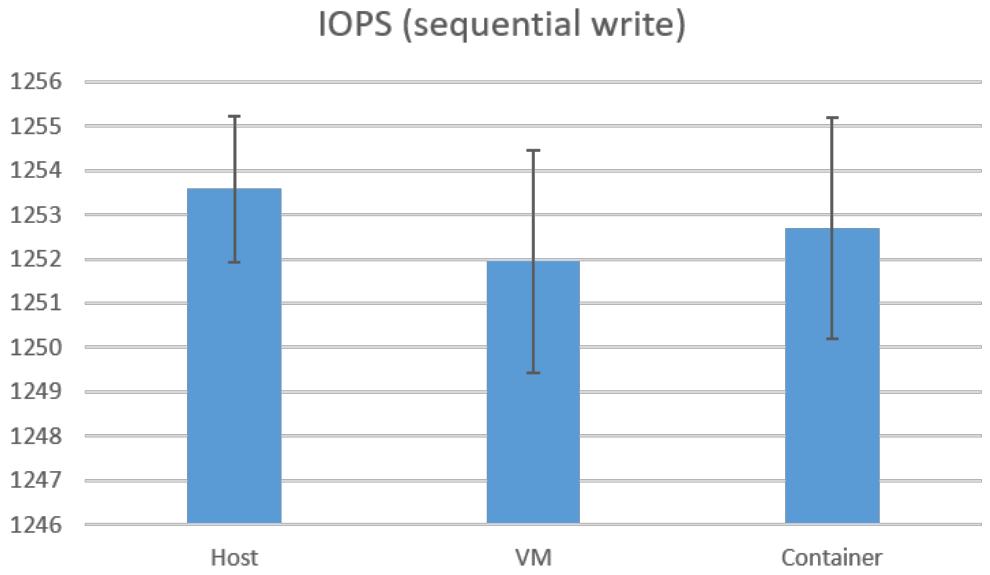
Kommando for lesedelen:

```
diskspd.exe -c5G -d120 -s -w0 -t1 -o16 -b64k -h -L -Z1G g:\TestFile1.dat
```



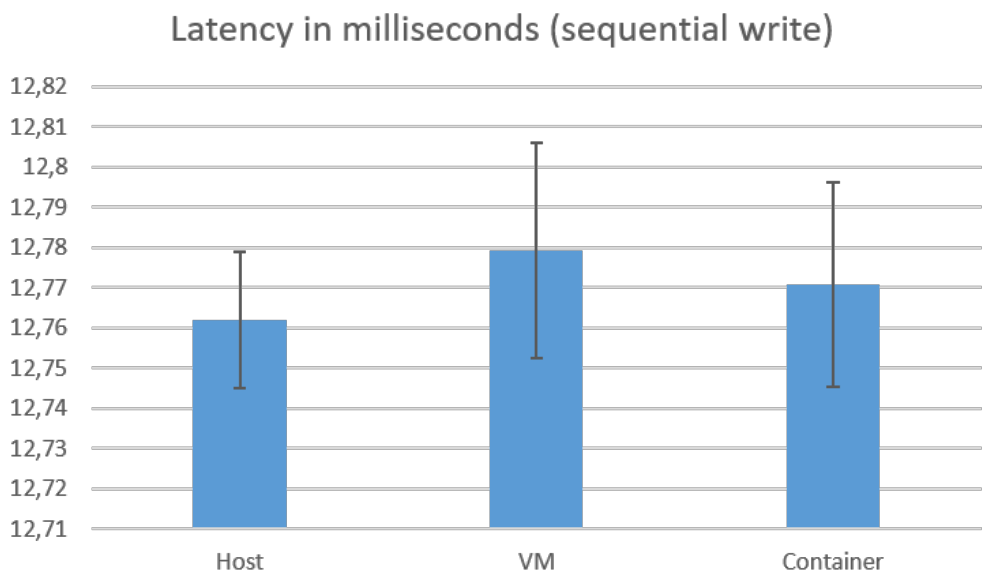
Figur 52: Gjennomstrømming på skriving sekvensielt i filen, målt i MB/s.

Figur 52 viser gjennomstrømming på skriving sekvensielt i filen. Host ligger øverst med en gjennomstrømming på 78,34 MB/s, men for første gang i ytelsestesten er container i midten med 78,30 MB/s. VM er den svakeste med en gjennomstrømming på 77,98 MB/s. VM har det største standardavviket med 0,61.



Figur 53: IOPS på skriving sekvensielt i filen.

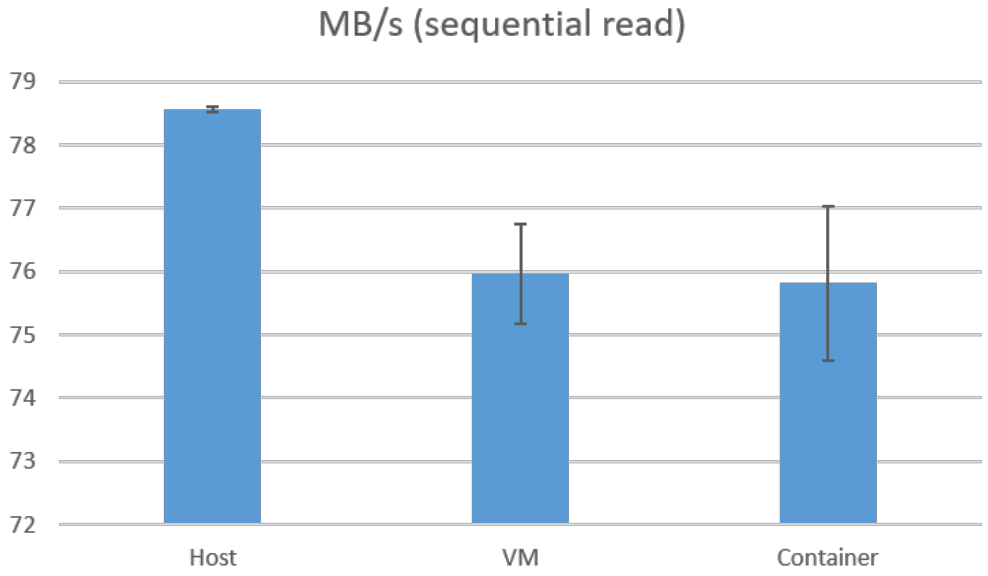
På samme måte som på gjennomstrømming er host og container svært like i ytelse med henholdsvis 1253,57 IOPS og 1252,70 IOPS. VM ligger nederst med 1247,63 IOPS (figur 53).



Figur 54: Latency på skriving sekvensielt i filen, målt i millisekunder.

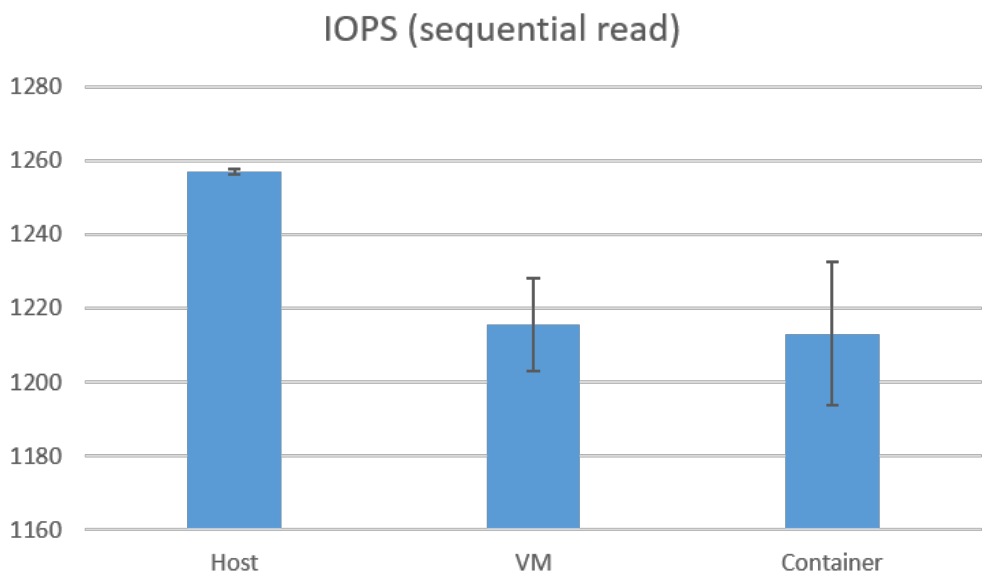
Host har laveste latency med 12,76ms, container i midten med 12,77ms. VM har høyest latency med 12,82ms. Med andre ord er forholdet mellom enhetene den samme

som i de to foregående målingene (figur 54). Årsaken til en lav latency er at det leses sekvensielt og det er ikke behov å vente på en hel rotasjon fra disken, sett i forhold til tilfeldig dataaksessering.



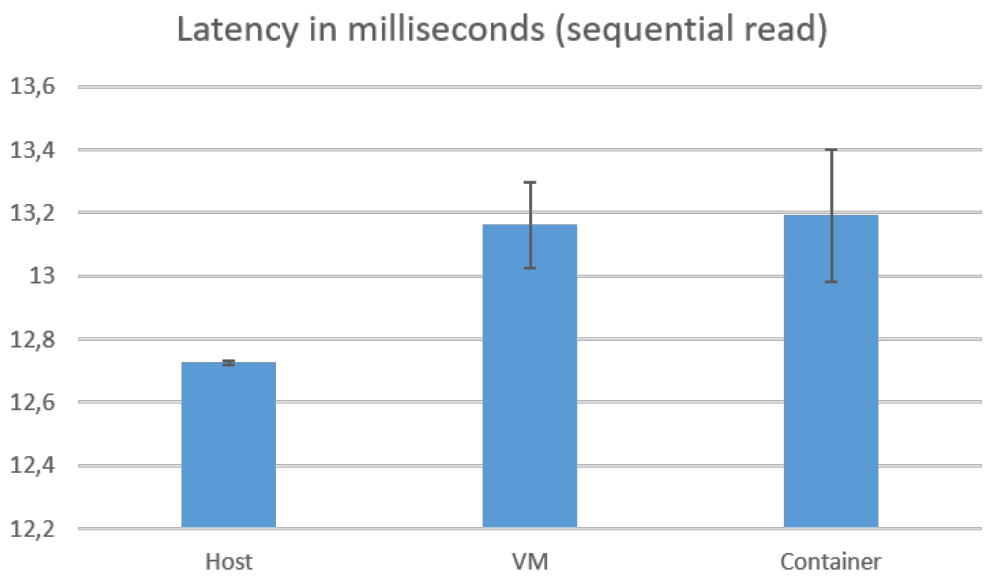
Figur 55: Gjennomstrømming på lesing sekvensielt i filen, målt i MB/s.

Figur 55 viser at VM og container er svært jevne på gjennomstrømming, men at VM er en anelse bedre. Host, VM og container har henholdsvis 78,57 MB/s, 75,97 MB/s og 75,82 MB/s.



Figur 56: IOPS på lesing sekvensielt i filen.

På samme måte som på gjennomstrømming er VM og container svært like med ca 2 IOPS i forskjell. Host, VM og container har henholdsvis 1257,03 IOPS, 1215,54 IOPS og 1213,13 IOPS (figur 56).



Figur 57: Latency på lesing sekvensielt i filen, målt i millisekunder.

På sekvensiell latency er forholdet mellom enhetene uendret i forhold til IOPS-målingen. Host, VM og container har henholdsvis 12,73ms, 13,16ms og 13,19ms. VM og container

er med andre ord svært like på latency (figur 57).

8.1.5 Nettverk

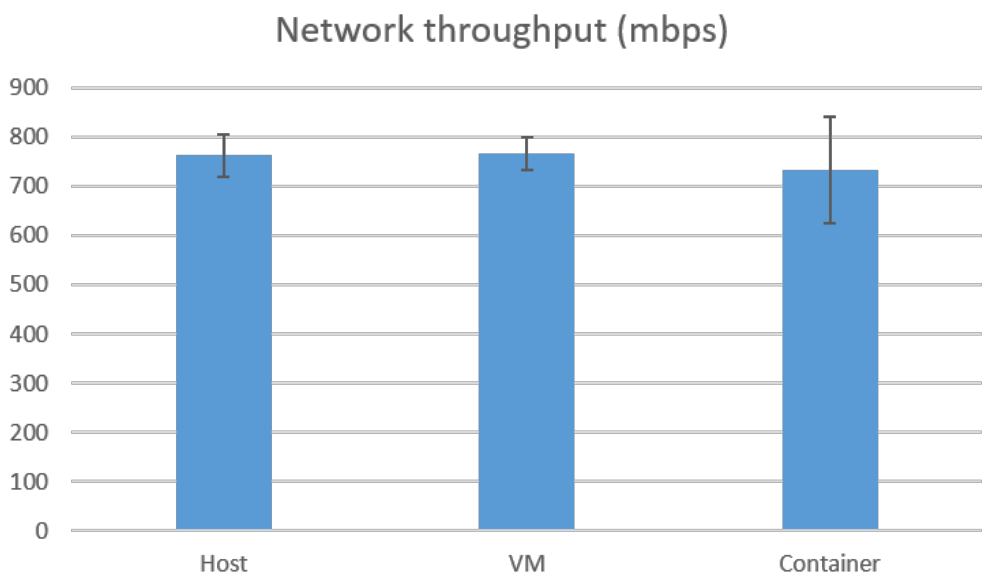
De ulike enhetene (host, VM, container) blir satt opp som sender *(-s)* med åtte tråder dynamisk fordelt over alle kjerner og sender data til mottakermaskinen *(-m 8,*, 192.168.0.2)* med 128K bufferlengde *(-l 128k)* og opererer i asynkron modus med to overlappende I/O buffere *(-a 2)* i 15 sekunder *(-t 15)* [76].

Kommandoen som ble kjørt på sender:

```
ntttcp.exe -s -m 8,*,192.168.0.2 -l 128k -a 2 -t 15
```

Kommandoen som ble kjørt på mottaker:

```
ntttcp.exe -r -m 8,*,192.168.0.2 -rb 2M -a 16 -t 15
```



Figur 58: Gjennomstrømming på nettverkstrafikk, målt i mbps

Host og VM yter tilnærmet likt på gjennomstrømming mens container ligger noe bak (figur 58). Container har i tillegg det desidert største standardavviket på 107,97 mbps og varierer fra en gjennomstrømming på 176,46 mbps til 791,64 mbps. VM har et standardavvik på 33,11 mbps og er med andre ord langt mer stabil. Host, VM og container har en gjennomstrømming på henholdsvis 762,35 mbps, 765,53 mbps og 732,14 mbps.

Del III

Avslutning

KAPITTEL

9

RESULTAT

9.1 Ytelse

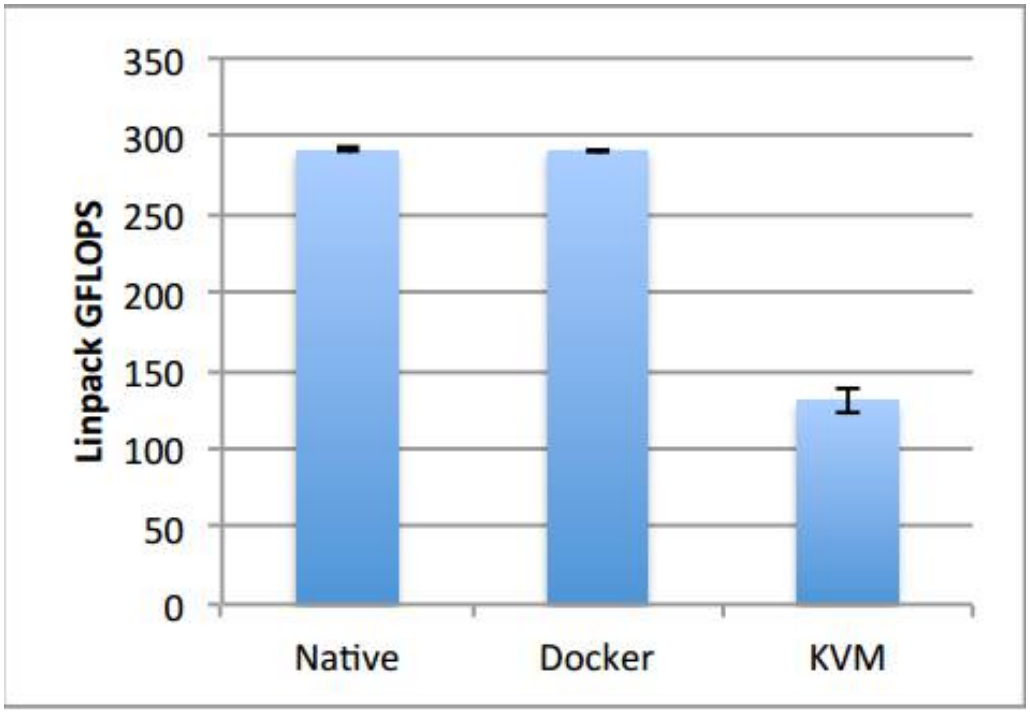
I så nær som samtlige tester kjørt med Geekbench kom containere ut som svakeste enhet (delkapittel 8.1.3), med tester av minnebåndbredde som eneste unntak, men her med et høyt standardavvik. Windows Container er tilsynelatende dårlig optimalisert og har en dårlig ressurshåndtering satt opp i mot en virtuell maskin. Dette var forventet i testen uten last, da det blir produsert en viss last ved at container host kjører en container, men ikke det forventede utfallet i test med last. Standardavviket for container var generelt sett høyt i testene noe som tyder på mer overhead ved at en virtuell maskin fungerer som container host.

I disktestene er tilstanden noe bedre (delkapittel 8.1.4). Container er svakeste enhet i samtlige tester på tilfeldig I/O, men forskjellene er små sett i forhold til VM (for eksempel IOPS hvor VM og container henholdsvis ligger på 167 og 165,5 IOPS). I testene på sekvensiell I/O yter container bedre enn VM på skrivedelen, men også her er marginene små (container hadde 1252,70 IOPS mot VM sine 1247,64 IOPS). I tester gjort av IBM på Linux er Docker og VM lik i ytelse på sekvensiell testing, mens på tilfeldig testing gir Docker betraktelig bedre resultater [3].

I testen på nettverksytelse hadde container et oppsiktsvekkende høyt standardavvik på 107,97 mbps og varierte fra en gjennomstrømming på 791,64 mbps ned til 176,46 mbps (delkappittel 8.1.5). Virtuelle maskiner virker bedre optimalisert på dette punktet og ligger tilnærmet likt med resultatene til host. I gjennomsnitt har container en 33,4 mbps lavere gjennomstrømming sett i forhold til VM.

Den eneste testen hvor containere kom ut som en klar vinner var ved testing av oppstarts- og stopptider (delkappittel 8.1.2). Her hadde VM en oppstartstid som var ~ 5 ganger så høy som containere (17,47s mot 3,5s). Dette var som forventet, ettersom containerne ikke trenger å kjøre opp systemprosesser.

Gruppen mener at årsaken til den svake ytelsen i stor grad kan skyldes at container host er nødt å være i en virtuell maskin i gjeldende technical preview av operativsystemet. Vi tror at denne dårlige optimaliseringen vil bli løst i nyere versjoner av operativsystemet. Dette er basert på testing gjort av IBM, der Linux container blir satt opp i mot Linux VM [3], i tillegg til at Microsoft har informert om at technical preview 5 kommer til å fjerne behovet for å kjøre container host i en VM. Som figur 59 viser, er Docker et stykke foran virtuelle maskiner i denne syntetiske testen.



Figur 59: Syntetisk ytelsestest på 16 kjerner gjort av IBM [3]

9.2 Egnede og uegnede bruksområder

I dette delkapittelet er det tatt forbehold om at technical preview 5 og videre versjoner av Windows Server 2016 løser den dårlige ressurshåndteringen og den svake ytelsen av containere.

Gruppen ønsker her å se nærmere på noen bruksområder for containere og deretter vurdere hvor vidt de egner seg i container eller på en virtuell maskin.

9.2.1 Programvareutvikling

Containere egner seg godt for programvareutviklere som kan få alle avhengighetene koden deres trenger raskt, og automatisk, satt opp ved hjelp av en dockerfil [82]. Når alle avhengighetene er på plass, koden er ferdigutviklet og pakket til et image, er det raskt å sette det ut i et testmiljø. For applikasjoner som består av mikrotjenester vil det være enda mer effektivt å teste ut funksjonalitet gjennom utviklingsprosessen, da det kan være komponenter av en applikasjon som må skaleres opp for å testes. Til slutt er det enkelt å sette en applikasjon ut i produksjon av en driftsorganisasjon som ikke må ta hensyn til applikasjonavhengigheter. ettersom disse kommer på plass fra imaget. Mer informasjon om dette finnes i delkapittel 9.3.

9.2.2 Webserver

Det finnes en hel del image til web servere fra Docker Hub, for eksempel Nginx som ligger øverst i Docker Hub [23]. Slik at det er enkelt å finne en type web server med den funksjonalitet du er ute etter. Docker Hub er beskrevet mer i detalj i delkapittel 3.1.2

En web server har en lastbegrensning på hvor mange klienter den kan betjene, og

antall forespørsler den kan behandle per sekund [83]. Om en web server blir overbelastet vil den slutte å svare. Dette løses med skalering, og dette passer containere godt til da de er enkle å skalere opp. I og med at de har en rask oppstartstid i forhold til virtuelle maskiner vil det også gå raskere å skalere opp når en nettside tar imot for mye trafikk og det blir behov for flere web servere. I et tilfelle hvor en stor mengde klienter kobler seg til en nettside over et kort tidsintervall kan containere ha fordel av at de klarer å skalere opp web servere fortere enn det virtuelle maskiner ville ha klart og man vil lettere unngå at nettsiden blir overbelastet. Det viktige er at antall web servere står i forhold til trafikken, slik at man ikke kjører unødvendig mange web servere.

Webservere inneholder ofte lite sårbar informasjon i seg selv, og det er derfor ikke den største trusselen om man mister innholdet eller kontrollen på en container som kjører web server. Den største trusselen oppstår om en angriper klarer å få kontroll over kernel som deles av alle containere til hosten. De fleste kjente sikkerhetstrusler appellerer både til containere og virtuelle maskiner (som for eksempel SQL Injection) [84].

9.2.3 Spillserver

Spillservere er et annet eksempel på en applikasjon som passer bra i en container. Dersom en tar utgangspunkt i det populære first-person shooterspillet “Counter Strike” vil en raskt se at det finnes tusenvis av servere med identiske innstillinger. Mange av disse er kjørt av skaperen av spillet, Valve. Spill-serverne oppdateres som regel i en tickrate på 64, altså den oppdaterer posisjonen til alle objekter i spillet 64 ganger i sekundet mens alt grafisk blir gjort på de enkelte klientene, og er derfor en ganske “lett” løsning for serveren. Dette i tillegg til det homogene servermiljøet skaper gode forhold for å bli implementert i containere.

I likhet med web servere støtter en spillservere bare et gitt antall forespørsler innenfor et gitt tidsrom (her begrenset av antall spillere tillatt på samme spillserver). Dette medfører at automatisk oppsett av spillservere når behovet inntreffer vil være en særdeles gunstig løsning.

Det omdiskuterte sikkerhetsaspektet med containere er i liten grad vesentlig innenfor slike spillservere da det er ingen sårbar informasjon på serverne [85]. Faren er på samme måte som i web servere om noen får kontroll over kernel og fjerner/hindrer alle serverne og skaper nedetid.

9.2.4 Komplette økonomisystem

Uni Micro leverer et komplett lønnsystem med blant annet fakturering og oppfølging, handel, logistikk, CRM, prosjektstyring, regnskap, lønn for personell og timeføring [86] [87]. Bortsett fra en SQL-database er dette et selvstendige program. Økonomisystemer er ofte monolittiske applikasjoner, det vil si at de er applikasjoner som er selvforsynt med de funksjonene den trenger og ikke er avhengige av andre applikasjoner [88]. Monolittiske applikasjoner har ikke det samme behovet for skalering og rask oppsettelse som mikro-tjenester. Derfor vil containere være mindre lønnsomt for monolittiske applikasjoner.

Ved å kjøre monolittiske applikasjoner i en virtuell maskin slipper man å risikere at en kernel-feil eller angrep kan ta ned andre monolittiske applikasjoner i andre virtuelle maskiner. Det er kritisk for en bedrift å miste oppetid for hele sitt økonomisystem eller i verste fall miste sensitiv informasjon, det er derfor ikke noe poeng i å gå vekk i fra den ekstra sikkerheten som en virtuell maskin kan tilby når det er lite å hente fra skalering

og effektivitet.

9.3 Når skal virtuelle maskiner brukes istedenfor Containere?

Det er en del store bedrifter, blant annet Google, IBM og Microsoft som har tatt i bruk containerteknologi, men dette betyr ikke at virtuelle maskiner er utdatert.

Containere gjør det mulig å kjøre flere enkle applikasjoner på samme fysiske server enn virtuelle maskiner kan. På dette bruksområdet er det mer optimalt å bruke containerteknologi, i hvert fall i teorien. I gjeldende versjon av Windows Server 2016 (technical preview 4) er det ikke mulig å opprette containere uten å sette opp en virtuell maskin som container host [21]. Dette er en begrensning som innfører et bestemt ytelsestap og økt overhead, noe som blir demonstrert i ytelsestestingen, kapittel 8.

Virtuelle maskiner benytter en relativt stor mengde systemressurser, ettersom det kreves en fullstendig versjon av et operativsystem i tillegg til hardwarevirtualisering, selv om det kun skal kjøres èn applikasjon på maskinen. Dette tar opp mye RAM og CPU-ytelse. Containere deler operativsystem (derav det alternative navnet operativsystem-virtualisering) og biblioteker, noe som i teorien sparer inn mye av disse ressursene.

I teorien kan det kjøres 2-3 ganger flere applikasjoner på en server med containere enn med virtuelle maskiner, men på grunn av at containere må kjøre i en virtuell maskin i gjeldende versjon av operativsystemet, fungerer ikke dette i praksis.

Slik som containere fungerer i technical preview 4 er det svært få tilfeller hvor det vil være bedre å kjøre containere enn virtuelle maskiner. I technical preview 5 skal det bli mulig å kjøre containere med den fysiske serveren som container host, noe som trolig vil forbedre ytelsesresultatene til containere.

I Windows Server 2016 - technical preview 4 er containere på ingen måte en konkurrent og i beste fall en komplementær teknologi til virtuelle maskiner. Fordelene med oppstartstid, diskbesparelse og muligheten til å håndtere images, er i stor grad utveid av ulempene som mangel på GUI, tilsynelatende dårlig ressurshåndtering, og svak ytelse.

Dersom en tar utgangspunkt i at technical preview 5 og/eller videre versjoner løser ytelsesproblemene, vil det være riktig å bruke containere til fordel for virtuelle maskiner til å kjøre enkle applikasjoner, spesielt om det er behov for horisontal skalering på disse applikasjonene, for eksempel web -og spillservere. Containere vil være riktig sammen med mikroarkitektur der en større applikasjon er satt sammen av flere mindre og enkle applikasjoner. I tillegg er containere et godt valg for utvikling, da det letter arbeidet ved å kunne pakke inn avhengighetene i en dockerfil og enkelt sette opp testmiljø. Virtuelle maskiner bør brukes dersom det skal kjøres ulike, flere eller store applikasjoner, da dette er noe containere er spesielt egnet til [89]. Dersom sikkerhet er høyt prioritet og det er vitalt at applikasjoner er isolert fra hverandre, bør en også bruke virtuelle maskiner.

KAPITTEL

10

EVALUERING

10.1 Innledning

Ettersom ingen i gruppen har jobbet på et prosjekt av denne størrelsen før, valgte gruppen å samles i et grupperom hvor arbeidet fant sted. Dette ble gjort for å gjøre det lettere å komme med innspill til hverandre og støtte opp med nødvendig kunnskap der dette skulle være nødvendig. Det har vært et bra gruppesamarbeid med ingen eller få store uenigheter og en god arbeidsvilje.

10.2 Organisering

Organisering i gruppen har blitt beskrevet i vedlegg A del A.8. Dette har fungert bra, alle har jobbet med oppgaver fordelt fortløpende i prosjektet. Gruppen har hatt god kontakt med veileder gjennom hele utførelsen av oppgaven vha. ukentlige statusmøter og tilbakemelding. Dette har gjort fremdriften i prosjektet mer effektiv, i tillegg til å få en bedre forståelse av forventningene, slik at sluttresultatet blir tilfredsstillende. Gruppen har i tillegg hatt kontakt med oppdragsgiver via e-post og møter når dette var nødvendig.

10.3 Gruppearbeid

Gruppen har oppholdt seg på et felles grupperom for gjennomføring av arbeid gjennom hele oppgaven. Dette har gitt gruppen en veldig god kommunikasjonsflyt mellom gruppemedlemmene og forenklet samarbeidsprosessen. Gruppen har tidligere jobbet sammen på andre prosjekter og medlemmene var godt kjent med hverandre før starten av oppgaven, noe som gjorde oppstarten av oppgaven lettere. Når større problemer oppstod ble disse diskutert i fellesskap innad i gruppen dersom dette viste seg å være nødvendig. Arbeid som skulle gjennomføres ble skrevet i et dokument i Google Docs, og fordelt som beskrevet i neste punkt.

10.4 Fordeling av arbeid

Det har ofte dukket opp nye problemstillinger som må løses, og disse har blitt fordelt fortløpende etter ønske og kompetanse. Ved at noen i gruppen hadde bedre kompetanse på visse områder har dette vært tidseffektivt. Ellers har arbeidet blitt fordelt likt mellom medlemmene, dette har blitt gjort av gruppeleder dersom ingen tok på seg ansvaret for en aktuell oppgave.

10.5 Prosjekt som arbeidsform

Gruppen har mye erfaring med prosjekt som arbeidsform fra tidligere oppgaver ved NTNU, blant annet i fagene systemutvikling, risikovurdering og hendelseshåndtering, men aldri med prosjekter av denne størrelsen. Oppgaven har gitt gruppen et godt innblikk i hvordan det er å jobbe med større prosjekter over lengre tid samtidig som en gjennomfører andre fag og gjøremål.

10.6 Gjennomføring av fremdriftsplan

Oppsett av arbeidsmiljø tok lenger tid enn forventet grunnet problemer med technical preview og det ble foretatt nye installasjoner av operativsystemet en rekke ganger, noe som medførte at serveren ikke var stabil før 24. februar (figur 61). Da både case 1 og case 2 ble forkastet 5 var denne delen av fremdriftsplanen vanskelig å følge og

derfor ble arbeidet med casene avsluttet 25. februar. Casene var i utgangspunktet en stor del av fremdriftsplanen og derfor ble den originale planen sett bort fra etter 25. februar (figur 61). Det ble derfor formulert en ny plan der det i første omgang ble sett grundigere på arkitektur, teori og bruksområder av containere. Deretter ble det gjort en gjennomgang av begrensninger og fordeler med en implementasjon av Docker, blant annet ved utførelse av ytelsestester.

10.7 Hva kunne vært gjort bedre

I starten av prosjektet brukte vi mye tid på å sette opp og beskrive en VDI-løsning på en virtuell maskin først, med hensikt å deretter kjøre ytelsestester mot en lignende løsning i containere. Ettersom Windows Container viste seg å ikke ha støtte for et skrivebord, var dette en umulig oppgave. Dette gikk for lang tid før dette ble oppdaget. I tillegg ble det sluppet en video 3. mars [2] hvor Microsoft forteller hvordan RDP mest sannsynlig ikke blir støttet i retailversjonen av Windows Server 2016. Gruppen hadde gått ut fra at dette bare var en mangel i technical preview 4, men dette viste definitivt at en VDI-løsning ikke lot seg gjøre. Før denne videoen ble sluppet, ble mye arbeid satt til side mens gruppen ventet på technical preview 5 i håp om at denne skulle muliggjøre casene. Gruppen skulle heller ha brukt tiden på å utarbeide en alternativ løsning.

Casene som gruppen utarbeidet i samarbeid med oppdragsgiver og veileder baserte seg i stor grad på at en VDI-løsning var gjennomførbar. Da dette viste seg å ikke være mulig, var gruppen nødt til å gå bort fra arbeidet med det og vinkle oppgaven på en ny måte. Dette kostet mye tid, selv om deler av dette var uunngåelig. Dersom det hadde blitt lagt mer forarbeid i å undersøke hvilke tjenester som eksisterte i technical preview 4 og Windows Server 2016, kunne casene vært gjennomførbare.

På tidspunktet da demoen ble laget for Ikomm var det mest aktuelt å bruke PowerShell-kommandoer for å produsere et resultat. Dette var på grunn av at Docker-kommandoene ikke fungerte som tiltenkt og i stor grad var preget av bugs. I ettertid er dette blitt løst ved lanseringen av technical preview 5. Det hadde vært ønskelig å gjennomført denne demoen med Docker-kommandoer dersom technical preview 5 hadde blitt lansert tidligere.

Det ville vært ønskelig med mer informasjon om hvordan filsystemet på Windows Container fungerer, men det er svært lite informasjon om oppbygningen av filsystemet tilgjengelig for øyeblikket.

På grunn av mangler i containere på Windows Server 2016, og lite utvalg i software, var det ikke gjennomførbart å teste random access (GUPS). Ettersom dette er noe som er viktig i et servermiljø, hadde vi helst villet testet dette.

Da denne oppgaven omhandlet programvare som fortsatt er i technical preview var gruppen forberedt på at det kom til å bli en del feil, men det skulle vært lagt mer tid i å undersøke hvilke begrensninger dette medførte før arbeidet med blant annet VDI-løsning ble påbegynt.

Risikovurderingen gruppen gjorde angående at Ikomm ikke kunne bistå med uventede problemer (vedlegg A.10.2) fikk høy sannsynlighet på å inntreffe ettersom vi visste Ikomm ikke hadde sett på denne teknologien tidligere, og sannsynligvis ikke hadde tid til å se på eventuelle problemer. Dette momentet inntraff når gruppen fant ut av VDI ikke ville være støttet i technical preview 4 (delkapittel 5.1.2). Dette problemet førte til at casene ikke var gjennomførbare, og verken Ikomm eller veileder kunne bistå med dette

problemet, ettersom det var en grunnleggende mangel i teknologien noe som ikke kunne omgås.

Det oppstod enkelte problemer med technical preview 4 som gjorde at det var nødvendig å reinstallere operativsystemet til serveren flere ganger, dette førte til tap av arbeid. Virkningene av dette ble dempet av tiltaket som ble iverksatt på dette punktet. Gruppen lagret viktige bilder og dokumenter både fysisk og i skytjenester, så ingen data ble tapt ved problemene. Det førte dog til en del tap av tid ettersom serveren måtte konfigureres på nytt.

Gruppen burde hatt en risikovurdering på bruk av Windows Server 2016 - Technical Preview 4. Gruppen hadde lite forkunnskaper på hva det vil si å jobbe i en technical preview og eventuelle problemer som kunne medfølge. Det burde blitt gjort et grundigere forarbeid på mangler og feil i technical preview slik at risikovurderingen bedre kunne speile virkeligheten.

10.8 Hva har gruppen lært

Gruppen sitter igjen med en bredere kompetanse innenfor virtualisering generelt, containerteknologi, spesielt på Windows, og hvilke fordeler og avgrensninger denne teknologien har. Gruppen har i tillegg fått oppleve å jobbe med et større prosjekt, noe som har vært en svært nyttig erfaring.

10.9 Mulig videre arbeid

Ved utgivelsen av fullversjonen av Windows Server 2016 vil det være nyttig å kjøre benchmarks på nytt for å kunne se ytelsesforskjellene i forhold til nåværende versjon, mot virtuelle maskiner og et tilsvarende oppsett med Docker på Linux. Dette vil trolig gi et mer korrekt resultat ettersom containere vil kunne bli kjørt direkte på maskinvare og ikke igjennom en virtuell maskin.

KAPITTEL 11 KONKLUSJON

11.1 Spørsmålene fra Ikomm

I oppgavebeskrivelsen kom Ikomm med en rekke spørsmål,delkapittel 1.1. I denne delen vil vi etter beste evne forsøke å besvare disse.

11.1.1 Hvordan vil Docker påvirke disse plattformene (VmWare, HyperV og Xenserver)?

Gruppen tolker dette spørsmålet som "Hvilke forskjeller vil det være i bruken av containere på de ulike plattformene?". Ikomm har sagt at de ønsker å gå fra en kombinert løsning med VmWare, XenServer og Hyper-V, til primært Hyper-V. Basert på denne beslutningen har oppgaven fokusert på Hyper-V, som er integrert i Windows Server. Det er derfor ikke sett på hvilken innflytelse Docker har på andre hypervisorer. For å svare på dette spørsmålet, blir det likevel gitt en kort oppsummering av containerteknologier på VmWare og XenServer. Begge to har utviklet Docker-integrasjon for deres respektive plattform.

I XenServer er det støtte for Docker tilgjengelig i XenServer 6.5 SP1 [90]. Programvaren XenServer leverer for container-administrasjon, kalles XenCenter. XenCenter har et brukergrensesnitt med følgende funksjonalitet:

- Monitorering og synlighet: Se hvilke VMer som brukes til Docker og hvilke containere som kjøres på disse.
- Diagnostikk: Vise grunnleggende containerinformasjon, for eksempel åpne nettverksporter og navn på Docker image.
- Ytelse: Viser hvilke containere som kjører på en virtuell maskin, hvilke prosesser som kjører i containerne og hvor mye CPU-tid hver prosess bruker.
- Kontrollere applikasjoner: Starte, stoppe og sette containere på pause.

(Liste hentet fra [90])

VMware har utviklet programvaren "vSphere Integrated Container"(VIC) som knytter Docker daemon til en vSphere-sky [91]. VIC består av flere komponenter, blant annet en web-klient for å styre og overvåke containere [92]. Mer informasjon om Docker på VMware og XenServer finnes på referansene i dette delkapittelet.

Nested virtualization er en teknologi som gjør det mulig å kjøre en eller flere Hyper-V servere i en VM. Dette er viktig for implementasjonen av Hyper-V Container, ettersom det tillater å kjøre slike containere i en virtuell maskin [93]. Ulikt fra Windows Container, kjører Hyper-V Container på toppen av en Hyper-V hypervisor, se kapittel 3.3. Nested virtualization gjør det mulig å lage Hyper-V containere i en virtuell maskin som kjører Windows Server 2016.

11.1.2 Er Docker en komplementær teknologi, eller en konkurrent til nåværende plattform?

Containere og virtuelle maskiner tilbyr ulik grad av isolasjon. Mens containere gir prosess-isolering på OS-nivå, gir virtuelle maskiner isolering av maskinvarelaget. Virtuelle maskiner passer bedre i IaaS (Infrastructure as a service) hvor man blant annet tilbyr VDI-løsninger, som ikke er mulig med Docker på Windows. Containere er i teorien bedre til å pakke og shippe programvare, og skalering av mikrotjenester, se delkapittel 3.4.5 og 9.3.

Operativsystem-avhengighet

Virtuelle maskiner kan brukes til å sette opp forskjellige operativsystemer på en vert, i motsetning til containere som kjører samme operativsystem som verten. Uten denne typen virtualisering må man dedikere hele verten til Windows Server 2016. Det er derfor behov for virtuelle maskiner når tjenester som er avhengig av ulike operativsystem skal kjøre på samme fysiske server

Sikkerhetskrav

Noen applikasjoner ønsker man å kjøre i et så sikkert miljø som mulig. Containere som deler kernel med verten er mindre sikre enn VMer. Disse burde plasseres i Hyper-V Containere som gir applikasjonen sin egen kernel, eller i en virtuell maskin. En mengde containere kan alltids isoleres mot den fysiske verten ved å sette de opp på en VM.

Skalering

Windows Container har i teorien en stor fordel med tanke på den minimale ressursbruken per container. Ut fra ytelsestestene som er kjørt av gruppen, har ikke gjeldende versjon av Windows Container bedre ressursutnyttelse sett i forhold til virtuelle maskiner, noe som medfører at oppstartstid, minnebruk og diskplass blir de gjenværende fordelene med containere med tanke på skalering.

I skrivende stund vil ikke Docker på Windows kunne konkurrere med nåværende platform, da det er flere vitale deler, spesielt mangelen på et grafisk grensesnitt og RDP-tilkobling som gjør til at den ordinære virtuelle maskinen fortsatt vil være den ledende teknologien på dette stadiet. Se detaljert beskrivelse angående RDP-støtte i kapittel 5.1.2. Hvorvidt der er verdt å kjøre en applikasjon i containere på Windows er noe som må vurderes fra tilfelle til tilfelle. Krever den grafisk grensesnitt, er det i stor grad utelukket, selv om det kun er for installasjon. Eksempler på tilfeller hvor Windows Container er egnet, kan være lette applikasjoner hvor rask oppstartstid er viktig, applikasjoner utviklet

i et DevOps miljø innad i en bedrift, og i tilfeller hvor gjerrig bruk av diskplass er viktig.

11.1.3 Finnes det referanseprosjekter på bruk av Docker som er relevante for Ikomm?

Det er flere organisasjoner som distribuerer sine applikasjoner og tjenester med Docker-containere som kjører på Linux. Det er svært få av disse som er direkte sammenlignbare med Ikomm. Et par selskaper har skrevet om sine erfaringer med denne teknologien, som kan være relevant, selv om selskapene i seg selv ikke driver innenfor samme bransje. Nå som Windows Container kun eksisterer i en beta versjon er det vanskelig å finne referanseprosjekt på denne plattformen. Docker ble tilgjengelig tidligere på Linux enn Windows og det er tydelig at Linux er mer utbredt som plattform ute i produksjonsmiljø. Shopify er en organisasjon som har brukt containere til å drifte over 100 000 nettbutikker [94]. De har beskrevet hvordan de gikk frem for å få til et slikt omfattende produksjonmiljø. Det som kan være relevant er deres fokus på å lage tynne containere som bare har nødvendig innhold for å forbruke minst mulig CPU og minne. De har også sentralisert tjenester til logging og innsamling av statistikk på hosten slik at de ikke ligger som duplikater på hver container.

En annen organisasjon, Riot Games Engineering, tar for seg implementering av Docker containere i en av Riot's største serverpark for å drifte spillet League of Legends. De følger fire grunnprinsipper i deres kontinuerlige leveranse av tjenester [95]:

- Utviklingsteamet må eie og ha full administrativ kontroll over deres byggemiljø.
- De ansatte burde vedlikeholde deres oppsett av pipeline og miljø i source control ved hjelp av konfigurasjonskode (dockerfil).
- Hver gang en utvikler lager programvare må den bygges kompatibel med alle konfigurerte miljøer.
- Å shippe kode er en produktavgjørelse. Produksjonsteam må ha muligheten til å enkelt gjøre tilgjengelig den nyeste versjonen av et produkt.

Docker ble den beste løsningen for å oppnå disse målene. Det første som forenklet leveranse var dockerfil som var enklere å vedlikeholde enn de tidligere verktøyene. Serverparken til Riot hadde over 3300 jobber og mange av disse var mikrotjenester. De prøvde i første omgang å kjøre rundt 500 jobber som de sparte mye ressurser på å kjøre i containere. Docker gav muligheten for å sette kode enkelt ut i produksjon. Som resultat opplevde de at utviklerne kunne bygge mikrotjenester og websider basert på Linux gjennom sine egne definerte miljøer, slik at utviklernes lokale byggemiljø fungerte sammen med miljøet i serverparken. For øvrig klarte ikke Docker å løse problemer i deres Windows og OSX miljøer, men tok et langt skritt i å løse utfordringer på Linux.

11.1.4 Hva slags fordeler kan Ikomm få ved å anvende denne type teknologi (gevinstrealisering)?

Slik gruppen vurderer det, vil ikke Ikomm ha noen nytte av å implementere Docker. Dette er på grunn av manglende støtte for VDI, og det at Ikomm selv har forklart, med at de har ikke mange applikasjoner med behov for horisontal skalering og det vil derfor bli ekstra unødvendig arbeid med å måtte håndtere et lite antall containere fra en container host i tillegg til de virtuelle maskinene. Som vist i gruppens ytelsestester (se kapittel 8) er ikke Windows Container ressursbesparende i gjeldende versjon, men dette vil trolig bli

løst i endelig versjon av Windows Server 2016. Uansett om det blir løst, vil de eventuelle ressursbesparelsene på ingen måte oppveie de organisatoriske ulempene.

11.1.5 Hvordan påvirker Docker leveransen av terminalservere/VDI?

Som forklart i delkapittel 5.1.2, ble det den 3.mars ble det sluppet en video [2] fra Microsoft der Taylor Brown forklarer at RDP ikke vil bli støttet med Windows Container. Dermed vil det ikke være mulig å levere VDI-løsninger på Windows Container.

11.1.6 Hvordan påvirker Docker leveransen av andre type applikasjoner som økonomisystemer, arkivsystemer o.l.?

Som forklart i delkapittel 5.1.2: Docker på Windows har ikke støtte for grafiske komponenter, noe som medfører at det ikke vil være mulig å kjøre applikasjoner som krever dette. Det er mulig å kjøre større applikasjoner som kan driftes uten skrivebord, men virtuelle maskiner vil i mange tilfeller fortsatt være bedre egnet til denne oppgaven. Som vist i 8, er dagens containere dårlige på å utnytte resursene de får tildelt.

11.1.7 Hvilke endringer i kompetansekrav kommer som en konsekvens av Docker teknologien?

Ved bruk av Docker vil man måtte ha en bredere kompetanse om hvordan de forskjellige virtualiseringsverktøyene fungerer. Det er viktig å være bevisst på hva som skiller containerteknologi fra ordinære virtuelle maskinene, og å ha kjennskap til når den ene eller den andre bør brukes. Nye tekniske ferdigheter ved implementering av containerteknologi, vil være eventuelt nye Powershell-moduler og Docker commands. Noen av disse er demonstrert i kapittel 4. Det vil også være nyttig å ha kjennskap til den omliggende infrastrukturen Docker og Microsoft tilbyr rundt denne teknologien, som er skrevet om i kapittel 6.

11.1.8 Hvor ligger utfordringene i bruken av Docker, både av organisatorisk og teknisk art?

Før en eventuell innføring av Docker i en organisasjon vil det være lurt å kjøre tester på forhånd, for å vite hvilke programmer som egner seg til å kjøre i containere, og hvor stor innsparing dette eventuelt fører til. Dersom en applikasjon krever et grafisk grensesnitt eller RDP-støtte er dagens Windows Container utelukket. Som demonstrert i kapittel 8 er det ingen garanti for at det er ressursbesparende å kjøre containere heller enn virtuelle maskiner. Ved innføring av Docker må det tas hensyn til at Windows container base image må være oppdatert til samme versjon som host-operativsystem som det er blitt forklart om i delkapittel 3.1.1, og det ikke mulighet for å oppdatere containere som allerede kjører.

BIBLIOGRAFI

[1] Tanenbaum, A. S. & Bos, H. 03 2014. *Modern Operating system Fourth Edition*. Pearson.

[2] Brown, T. & McSpirit, M. 03 2016. Day 3: Containers session with q&a. https://channel9.msdn.com/Events/TechNetVirtualConference/TechNetVC2016/Day-3-Containers-Session-with-QA. [Online; accessed 03-05-2016].

[3] Felter, W., Ferreira, A., Rajamony, R., & Rubio, J. 07 2014. An updated performance comparison of virtual machines and linux containers. http://www.cs.nyu.edu/courses/fall14/CSCI-GA.3033-010/vmVcontainers.pdf. [Online; accessed 08-05-2016].

[4] Vaughan-Nichols, S. J. 08 2014. What is docker and why is it so darn popular? http://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/. [Online; accessed 11-02-2016].

[5] Ikomm. 27 2016. Om ikomm. http://www.ikomm.no/om-oss/. [Online; accessed 28-01-2016].

[6] Bitner, B. 2012. z/vm – a brief review of its 40 year history. http://www.vm.ibm.com/vm40hist.pdf. [Online; accessed 22-02-2016].

[7] Rouse, M. 12 2010. Exploring data virtualization tools and technologies. http://searchservervirtualization.techtarget.com/definition/virtualization. [Online; accessed 06-05-2016].

[8] Rouse, M. 06 2009. Server virtualization. http://searchservervirtualization.techtarget.com/definition/server-virtualization. [Online; accessed 06-05-2016].

[9] Marshall, D. 11 2011. Virtualization report. http://www.infoworld.com/article/2621446/server-virtualization/server-virtualization-top-10-benefits-of-server-virtualization.html. [Online; accessed 06-05-2016].

[10] Fisher, S. 02 2013. 7 enterprise-wide benefits of virtualized servers. https://www.laserfiche.com/simplicity/7-enterprise-wide-benefits-virtualized-servers/. [Online; accessed 06-05-2016].

[11] Rouse, M. 10 2006. Hypervisor. http://searchservervirtualization.techtarget.com/definition/hypervisor. [Online; accessed 08-05-2016].

[12] Chenley. 02 2011. Hyper-v: Microkernelized or monolithic. https://blogs.technet.microsoft.com/chenley/2011/02/23/hyper-v-microkernelized-or-monolithic/. [Online; accessed 08-05-2016].

[13] Davis, D. 05 2013. The top 5 enterprise type 1 hypervisors you must know. http://www.virtualizationsoftware.com/top-5-enterprise-type-1-hypervisors/. [Online; accessed 08-05-2016].

[14] Rouse, M. 12 2014. Type 2 hypervisor (hosted hypervisor). http://searchservervirtualization.techtarget.com/definition/hosted-hypervisor-Type-2-hypervisor. [Online; accessed 08-05-2016].

[15] Hogg, S. 05 2014. Software containers: Used more frequently than most realize. http://www.networkworld.com/article/2226996/cisco-subnet/software-containers--used-more-frequently-than-most-realize.html. [Online; accessed 01-05-2016].

[16] Peterson, N. 03 2016. Windows containers. https://msdn.microsoft.com/en-us/virtualization/windowscontainers/about/about_overview. [Online; accessed 09-04-2016].

[17] Russinovich, M. 03 2016. Containers: Docker, windows and trends. https://azure.microsoft.com/nb-no/blog/containers-docker-windows-and-trends/?cdn=disable. [Online; accessed 13-04-2016].

[18] Opensource.com. 04 2016. Introduction to docker. https://opensource.com/resources/what-docker. [Online; accessed 21-04-2016].

[19] Ostrowski, R. 08 2015. Getting started with docker: Simplifying devops. https://www.toptal.com/devops/getting-started-with-docker-simplifying-devops. [Online; accessed 12-04-2016].

[20] Ziemke, J. 08 2014. How to handle security updates within docker containers. https://serverfault.com/questions/611082/how-to-handle-security-updates-within-docker-containers/. [Online; accessed 03-05-2016].

[21] Cooley, S. 02 2016. Work in progress. https://msdn.microsoft.com/en-us/virtualization/windowscontainers/about/work_in_progress. [Online; accessed 09-05-2016].

[22] Butler, T. 08 2014. Docker and windows server containers: A first look. https://www.conetix.com.au/blog/docker-and-windows-server-containers-first-look. [Online; accessed 03-05-2016].

[23] Docker. 04 2016. Overview of docker hub. https://docs.docker.com/docker-hub/overview/. [Online; accessed 21-04-2016].

[24] Friis, M. 04 2016. Docker on windows server 2016 technical preview 5. https://blog.docker.com/2016/04/docker-windows-server-tp5/. [Online; accessed 29-04-2016].

[25] Hassell, J. 01 2016. Windows server 2016 technical preview 4. http://www.computerworld.com/article/3024274/microsoft-windows/review-windows-server-2016-technical-preview-4.html. [Online; accessed 10-05-2016].

[26] Peterson, N. 02 2016. Windows containers quick start. https://web.archive.org/web/20160305155831/https://msdn.microsoft.com/en-us/virtualization/windowscontainers/quick_start/manage_powershell. [Online; accessed 05-03-2016].

[27] Weir, A. 08 2015. What's new in windows server 2016 technical preview 3. http://www.neowin.net/news/whats-new-in-windows-server-2016-technical-preview-3. [Online; accessed 13-04-2016].

[28] Savill, J. 08 2015. The differences between windows containers and hyper-v containers in windows server 2016. http://windowsitpro.com/windows-server-2016/differences-between-windows-containers-and-hyper-v-containers-windows-server-201. [Online; accessed 17-02-2016].

[29] Bigelow, S. J. 04 2016. Five cons of container technology. http://searchservervirtualization.techtarget.com/feature/Five-cons-of-container-technology. [Online; accessed 18-04-2016].

[30] Coleman, M. 03 2016. Containers are not vms. https://blog.docker.com/2016/03/containers-are-not-vms/. [Online; accessed 14-05-2016].

[31] Litt, S. 04 2014. Newbie's overview of docker. http://www.troubleshooters.com/linux/docker/docker_newbie.htm. [Online; accessed 10-04-2016].

[32] Mouat, A. 02 2016. Five security concerns when using docker. https://www.oreilly.com/ideas/five-security-concerns-when-using-docker. [Online; accessed 14-05-2016].

[33] Liyanage, Y. 05 2015. Docker - clean up after yourself! http://blog.yohanliyanage.com/2015/05/docker-clean-up-after-yourself/. [Online; accessed 14-04-2016].

[34] Copeland, B. 09 2015. Best practices for overcoming docker challenges. http://www.datacenterjournal.com/practices-overcoming-docker-challenges/. [Online; accessed 20-04-2016].

[35] Otto, A. 05 2015. Openstack magnum - containers-as-a-service. https://www.openstack.org/summit/tokyo-2015/videos/presentation/openstack-magnum-containers-as-a-service. [Online; accessed 10-05-2016].

[36] Otto, A. et al. 08 2015. Exploring opportunities:containers and openstack. https://www.openstack.org/assets/pdf-downloads/Containers-and-OpenStack.pdf. [Online; accessed 10-05-2016].

[37] Amor, P. 05 2015. Openstack and containers. http://blogs.cisco.com/cloud/openstack-and-containers. [Online; accessed 10-05-2016].

[38] Otto, A. 01 2015. Announcing magnum – caas for openstack. http://adrianotto.com/2015/01/announcing-magnum-caas-for-openstack/. [Online; accessed 10-05-2016].

[39] Shmuel Tyszberowicz, A. Y. 11 2007. Locating regression bugs. http://www.research.ibm.com/haifa/Workshops/verification2007/present/Dor_Nir_web.pdf. [Online; accessed 15-05-2016].

[40] Sendachi. 01 2016. How we use docker for continuous delivery. http://sendachi.com/2016/docker/how-we-use-docker-for-continuous-delivery-part-1/93. [Online; accessed 15-05-2016].

[41] Loukides, M. 06 2012. What is devops? http://radar.oreilly.com/2012/06/what-is-devops.html. [Online; accessed 14-05-2016].

[42] Oracle. 05 2016. Oracle ksplice. http://www.ksplice.com/. [Online; accessed 10-05-2016].

[43] Petazzoni, J. 08 2013. Containers & docker: How secure are they? https://blog.docker.com/2013/08/containers-docker-how-secure-are-they/. [Online; accessed 17-03-2016].

[44] Andersen, T. 05 2015. Live migration in lxd. https://insights.ubuntu.com/2015/05/06/live-migration-in-lxd/. [Online; accessed 12-05-2016].

[45] Warner, T. 07 2015. Microsoft nano server: Everything you need to know. https://www.pluralsight.com/blog/it-ops/microsoft-nano-server-announced. [Online; accessed 17-03-2016].

[46] PowerShell Team. 05 2016. xnetworking. https://github.com/PowerShell/xNetworking. [Online; accessed 15-05-2016].

[47] Citrix. 02 2016. What is vdi? https://www.citrix.no/glossary/vdi.html. [Online; accessed 22-02-2016].

[48] Tricerat. 02 2016. Virtual desktop infrastructure (vdi). http://tricerat.com/resources/topics-library/virtual-desktop-infrastructure-vdi. [Online; accessed 22-02-2016].

[49] Harbaugh, L. 03 2012. The pros and cons of using virtual desktop infrastructure. http://www.pcworld.com/article/252314/the_pros_and_cons_of_using_virtual_desktop_infrastructure.html. [Online; accessed 22-02-2016].

[50] Techtarget. 02 2007. terminal server. http://searchvirtualdesktop.techtarget.com/definition/terminal-server. [Online; accessed 22-02-2016].

[51] Cooley, S. 05 2016. Frequently asked questions. https://msdn.microsoft.com/en-us/virtualization/windowscontainers/about/faq. [Online; accessed 02-05-2016].

[52] Docker. 04 2016. Docker security. https://docs.docker.com/engine/security/security/. [Online; accessed 20-04-2016].

[53] Docker. 05 2016. Docker engine. https://www.docker.com/products/docker-engine. [Online; accessed 25-04-2016].

[54] Docker. 04 2016. Docker installation - windows. https://docs.docker.com/engine/installation/windows/. [Online; accessed 21-04-2016].

[55] Docker. 05 2016. Overview of docker compose. https://docs.docker.com/compose/overview/. [Online; accessed 06-05-2016].

[56] Docker. 05 2016. Docker compose. https://www.docker.com/products/docker-compose. [Online; accessed 21-04-2016].

[57] Techtarget. 04 2016. Docker swarm. http://searchcloudcomputing.techtarget.com/definition/Docker-Swarm. [Online; accessed 07-04-2016].

[58] Docker. 04 2016. Docker swarm overview. https://docs.docker.com/swarm/overview/. [Online; accessed 07-04-2016].

[59] Docker. 05 2016. High availability in docker swarm. https://docs.docker.com/swarm/multi-manager-setup/. [Online; accessed 07-04-2016].

[60] Research, I. 04 2016. Services computing. http://researcher.watson.ibm.com/researcher/view_group.php?id=152. [Online; accessed 21-04-2016].

[61] MSDN. 05 2016. Job objects. https://msdn.microsoft.com/en-us/library/windows/desktop/ms684161%28v=vs.85%29.aspx. [Online; accessed 21-04-2016].

[62] Hart, J. M. 02 2005. Windows processes and threads: Weaving it all together. http://www.informit.com/articles/article.aspx?p=362660&seqNum=15. [Online; accessed 21-04-2016].

[63] MSDN. 05 2016. Jobobject basic limit information structure. https://msdn.microsoft.com/en-us/library/windows/desktop/ms684147%28v=vs.85%29.aspx. [Online; accessed 21-04-2016].

[64] MSDN. 05 2016. Object namespaces. https://msdn.microsoft.com/en-us/library/windows/desktop/ms684295%28v=vs.85%29.aspx. [Online; accessed 25-04-2016].

[65] Messer, J. 02 2016. Container networking. https://msdn.microsoft.com/en-us/virtualization/windowscontainers/management/container_networking. [Online; accessed 05-05-2016].

[66] Marks, M. 01 2016. Unikernel systems joins docker. https://blog.docker.com/2016/01/unikernel/. [Online; accessed 24-04-2016].

[67] Microsoft Research. 05 2016. Drawbridge. http://research.microsoft.com/en-us/projects/drawbridge/. [Online; accessed 05-05-2016].

[68] Cormack, J. 04 2016. The road to unikernels. http://roadtounikernels.myriabit.com/?full#18. [Online; accessed 24-04-2016].

[69] Unikernel.org. 05 2016. Projects. http://unikernel.org/projects/. [Online; accessed 05-05-2016].

[70] Hammons, J. 04 2016. Windows subsystem for linux overview. https://blogs.msdn.microsoft.com/wsl/2016/04/22/windows-subsystem-for-linux-overview/. [Online; accessed 22-04-2016].

[71] Jacobs, J. 07 2008. Synthetic vs real world benchmarks. http://www.techwarelabs.com/articles/editorials/real-vs-synthetic/. [Online; accessed 05-05-2016].

[72] Primate Labs. 02 2016. Geekbench 3. http://www.primatelabs.com/geekbench/video/. [Online; accessed 04-05-2016].

[73] Adams, J. 07 2015. Diskspd utility: A robust storage testing tool (superseding sqlio). https://gallery.technet.microsoft.com/DiskSpd-a-robust-storage-6cd2f223. [Online; accessed 14-05-2016].

[74] Schulz, G. 02 2015. Server and storage i/o benchmark tools: Microsoft diskspd. http://storageioblog.com/server-storage-io-benchmarking-tools-microsoft-diskspd-part/. [Online; accessed 13-05-2016].

[75] Ajay.MSFT. 12 2015. Sqlio disk subsystem benchmark tool is being retired. https://blogs.msdn.microsoft.com/sql_server_team/sqlio-disk-subsystem-benchmark-tool-is-being-retired/#.Vm7uZvkrIyg. [Online; accessed 13-05-2016].

[76] Adams, J. 08 2015. Ntttcp utility: Profile and measure windows networking performance. https://gallery.technet.microsoft.com/NTttcp-Version-528-Now-f8b12769. [Online; accessed 14-05-2016].

[77] Talat, A. 05 2008. Nt... ttcp! network performance test tool available. https://blogs.technet.microsoft.com/winserverperformance/2008/05/03/nt-ttcp-network-performance-test-tool-available/. [Online; accessed 14-05-2016].

[78] Primate Labs. 05 2016. Interpreting geekbench 3 scores. http://support.primatelabs.com/kb/geekbench/interpreting-geekbench-3-scores. [Online; accessed 07-05-2016].

[79] Primate Labs. 05 2016. Geekbench 3 benchmarks. http://support.primatelabs.com/kb/geekbench/geekbench-3-benchmarks. [Online; accessed 07-05-2016].

[80] McCalpin, J. 01 2013. Stream benchmark. https://sites.utexas.edu/jdm4372/tag/stream-benchmark/. [Online; accessed 12-04-2016].

[81] Lowe, S. 01 2012. How do you measure virtual machine boot time? http://www.virtualizationadmin.com/blogs/lowe/news/how-do-you-measure-virtual-machine-boot-time-203.html. [Online; accessed 15-03-2016].

[82] Docker. 05 2016. Work with a development container. https://docs.docker.com/opensource/project/set-up-dev-env/. [Online; accessed 16-05-2016].

[83] Wikipedia. 2016. Web server — wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Web_server&oldid=719645788. [Online; accessed 16-May-2016].

[84] Hampton, T. J. 03 2011. 9 server security threats you should definitely know. http://www.webmasterview.com/2011/03/server-security-threats/. [Online; accessed 15-03-2016].

[85] Fosberry, B. 12 2015. How containers will change the game hosting industry. https://blog.codeship.com/how-containers-will-change-the-game-server-hosting-industry/. [Online; accessed 18-04-2016].

[86] Unimicro. 03 2016. unimicro. http://unimicro.no/. [Online; accessed 29-03-2016].

[87] Unimicro. 09 2013. Installasjon. http://support.unimicro.no/kundestotte/teknisk/installasjon-og-nettverk/installasjon/. [Online; accessed 29-03-2016].

[88] Wikipedia. 2014. Monolithic application — wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Monolithic_application&oldid=629481544. [Online; accessed 16-May-2016].

[89] Vaughan-Nichols, S. J. 04 2016. Containers vs. virtual machines: How to tell which is the right choice for your enterprise. http://www.itworld.com/article/2915530/virtualization/

containers-vs-virtual-machines-how-to-tell-which-is-the-right-choice-for-your-enterpris html. [Online; accessed 27-04-2016].

[90] Bulpin, J. 03 2015. Preview of xenserver support for docker and container management. http://xenserver.org/discuss-virtualization/virtualization-blog/entry/preview-of-xenserver-support-for-docker-and-container-management.html/. [Online; accessed 15-05-2016].

[91] Narayan, K. 04 2016. Introducing vsphere integrated containers open source software. http://blogs.vmware.com/cloudnative/introducing-vsphere-integrated-containers-open-source-software/. [Online; accessed 15-05-2016].

[92] Gray, E. 01 2015. vsphere integrated containers – technology walkthrough. http://blogs.vmware.com/vsphere/2015/10/vsphere-integrated-containers-technology-walkthrough.html/. [Online; accessed 15-05-2016].

[93] Syrewicze, A. 01 2016. How to enable nested virtualization on hyper-v & windows server 2016. http://www.altaro.com/hyper-v/nested-virtualization-hyper-v-windows-server-2016/. [Online; accessed 15-05-2016].

[94] Johnson, G. 11 2014. Docker at shopify: How we built containers that power over 100,000 online shops. https://engineering.shopify.com/17489060-docker-at-shopify-how-we-built-containers-that-power-over-100-000-online-shops [Online; accessed 08-05-2016].

[95] Stewart, M. F. 11 2014. Thinking inside the container. https://engineering.riotgames.com/news/thinking-inside-container. [Online; accessed 09-05-2016].

[96] Ikomm. 01 2016. Om ikomm - skytjenester. http://www.ikomm.no/om-oss/skytjenester. [Online; accessed 28-01-2016].

[97] Strauss, D. 06 2013. Containers—not virtual machines—are the future cloud. http://www.linuxjournal.com/content/containers%E2%80%94not-virtual-machines%E2%80%94are-future-cloud?page=0,1. [Online; accessed 11-02-2016].

[98] Wikipedia. 01 2016. Chroot wikipedia article. https://en.wikipedia.org/w/index.php?title=Chroot&oldid=712277422. [Online; accessed 28-march-2016].

[99] Wikipedia. 01 2016. Operating-system-level virtualization wikipedia article. https://en.wikipedia.org/w/index.php?title=Operating-system-level_virtualization&oldid=717879214. [Online; accessed 30-04-2016].

[100] Center, M. N. 10 2014. Docker and microsoft partner to bring container applications across platforms. https://news.microsoft.com/2014/10/15/DockerPR/. [Online; accessed 02-02-2016].

Del IV

Tillegg

TILLEGG

A

PROSJEKTPLAN

A.1 Oppdragsgiver

Ikomm AS er en IT-bedrift som holder til i Lillehammer med rundt 60 ansatte som jobber med å levere IKT-tjenester til privat og offentlig sektor. Tjenestene er blant annet konsulent- og rådgivningstjenester til andre som drift og på-stedet tjenester. Ikomm er dyktige til å tilpasse leveranse til kundens behov og drifter over 800 applikasjoner for mer enn 12000 brukere i Skandinavia. [5]

Kjernekompetansen til Ikomm er spesialisert mot fire hovedområder:

- Skytjenester
- Driftstjenester
- Rådgivning og prosjektstyring
- Klientservice

I 2015 ble Ikomm utvalgt som medlem av Cloud OS Network (COSN). Dette er en gruppe tjenesteleverandører på verdenbasis som samarbeider med Microsoft for å tilby kundene sine hybride skyløsninger, basert på Microsoft plattformen Azure.[96]

A.2 Bakgrunn for oppgaven

Ikomm bruker i dag primært virtualiseringteknologi til drifting av skytjenester på følgende plattformer:

- VmWare
- HyperV
- XenServer

De ønsker å finne ut hvordan de kan dra nytte av containerteknologien Docker og hvor velutviklet denne teknologien er på operativsystemet Windows Server 2016 til å anvendes sikkert og effektivt i deres sky.

A.3 Oppgavebeskrivelse

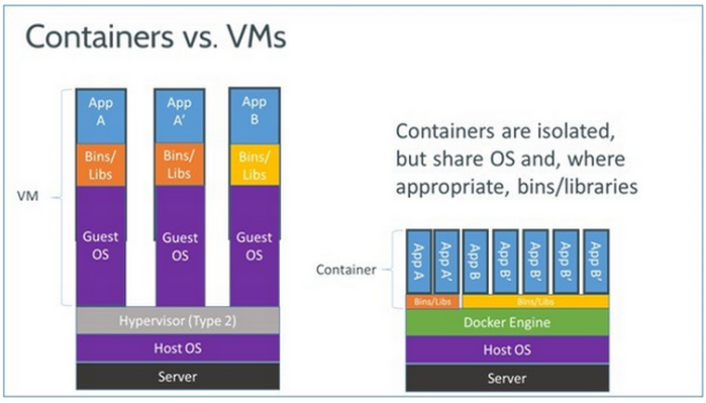
Ikomm er ute etter å få utredet hvordan containerteknologi som Docker vil påvirke organisasjoner som dem. Ikomm bruker i dag primært VmWare, HyperV og Xenserver som plattform som virtualisering, og bør ta utgangspunkt i følgende problemstillinger:

- Hvordan vil Docker påvirke denne plattformen?
- Er Docker en komplementær teknologi, eller en konkurrent til nåværende plattform?
- Finnes det referanseprosjekter på bruk av Docker som er relevante for Ikomm?
- Hva slags fordeler kan Ikomm få ved å anvende denne type tjenester (gevinstrealisering)?
- Hvordan påvirker Docker leveransen av terminalservere/VDI?
- Hvordan påvirker Docker leveransen av andre type applikasjoner som økonomisystemer, arkivsystemer o.l.?
- Hvordan endringer i kompetansekrav kommer som en konsekvens av Docker teknologien?
- Hvor ligger utfordringene i bruke av Docker, både av organisatorisk og teknisk art?

A.3.1 Hvilke fordeler kan Ikomm få av å implementere Docker ?

Applikasjoner som kjører på virtuelle maskiner er isolert av et helt gjesteoperativsystem, ofte på mer enn 1 GB, og er tungdrevet. De må kommunisere gjennom sitt gjesteoperativsystem når vertsoperativsystemet skal utføre input/output-operasjoner. Containere er små i størrelse og kommuniserer direkte med vertsoperativsystem. Forskjellen vil bety mindre overhead og vil da redusere ressursbruken av prosessorkraft, minne og harddiskplass for hver applikasjon som kjører på en fysisk server. I et datasenter vil dette gi muligheten til å kjøre flere applikasjoner på færre fysiske servere. Færre fysiske servere vil resultere i mindre strømutgifter for Ikomm.[97]

Det vil være ressurssparende å innkapsle applikasjoner i containere. Applikasjonene er isolert slik at applikasjoner ikke kan se eller påvirke hverandre når de kjører på en fysisk maskin (se fig. 60). Ved å ta i bruk Docker, kan datasentre spare store utgifter på maskinvare og strøm.



Figur 60: Containere vs. VM [4]

A.4 Vår bakgrunn

I gruppen studerer tre av oss Informasjonssikkerhet og en Drift av nettverk og datasystemer ved NTNU i Gjøvik. Ingen i gruppen har kunnskap om containerteknologi i forkant av prosjektet. De fagene vi har gjennomgått på universitetet som vil være relevante og nyttige for oss, er:

Systemadministrasjon: Det er tre av medlemmene som har fullført dette faget, dette faget vil komme godt med siden vi har fått erfaring med å sette opp virtuelle maskiner (VM) med blant annet image for Windows Server 2016.

Operativsystemer: Alle gruppemedlemmene har hatt gjennomgang i operativsystemer, som har gitt oss et innblikk i hvordan Windows er strukturert og designet i tillegg til grunnleggende scripting i Powershell.

Risikostyring: Tre av medlemmene har hatt risikostyring, dette vil hjelpe oss med å analysere risiko og hvordan det skal håndteres.

Database- og applikasjonsdrift: En av oss har tatt dette faget tidligere, og de tre andre tar nå faget parallelt med prosjektet. Dette vi komme til nytte i drifting av servere.

Systemutvikling: Dette vil komme til nytte under rapportskriving og gjennomføring av dette prosjektet.

A.5 Prosjektmål

A.5.1 Effektmål

Målet er at Ikomm skal ha et større kunnskapsgrunnlag å basere sine valg angående hvorvidt containerteknologi kan tas i bruk i deres daglige drift. Det ønskelige utfallet vil være at denne teknologien kan tas i bruk, og at denne skal effektivisere deres ressursbruk, i motsetning til en ren VM-basert drift.

A.5.2 Resultatmål

Hovedmålet med oppgaven vår er todelt:

1. Tilegne oss en gjennomgående kunnskap om containerteknologi, hvilke begrensinger og styrker den har i forhold til annen eksisterende teknologi, og da spesielt virtualisering. Emner som er aktuelle og som må evalueres i denne sammenheng har vi foreløpig vurdert til å være:
    - Kompatibilitet
    - Ressursbruk
    - Sikkerhet
    - Tekniske utfordringer
    - Kompetansekrav
    - Støttede applikasjoner
    - Effektivitet
    - Uavhengig eller komplementær teknologi?
2. Kunne demonstrere implementasjonen av Docker i et Windows-basert system.
    - Etter endt prosjekt skal vi kunne holde en presentasjon der vi demonstrerer ulike implementasjoner, i tillegg til hvilke brukstilfeller containerteknologi passer best.

A.6 Rammer

Prosjektet skal forholde seg til containerteknologien Docker som kommer inkludert i Windows Server 2016. Dette er er relevant for vår arbeidsgiver ettersom de er sterkt knyttet til Microsoft og har avtaler med dem som gir de visse goder, men også stiller krav til bruk av Microsoft-produkter. Operativsystem, maskinvare og programvare vil vi få utdelt av arbeidsgiver der vi vil ha vårt arbeidsmiljø. Her vil vi implementere og teste casene vi får tildelt av arbeidsgiver. Videre har prosjektet en tidsramme hvor rapporten skal være ferdig og levert innen 18. mai 2016.

A.7 Omfang

A.7.1 Fagområde

Vårt fagområde er virtualiseringsteknologi, og da spesielt containerteknologi. Containere er et relativt gammelt konsept. I 1982 kom "chroot" i Unix, som lager et separert filsystem for en applikasjon, slik at det ikke, kan aksessere det normale filsystemet på normal måte.[98] FreeBSD la til "FreeBSD jail" i 2000. Linux VServer kom i 2001.[99] Det som har endret seg i løpet av de siste par årene, er ikke så mye teknologien i seg selv, men hvor enkelt det har blitt å ta i bruk. Docker har ledet an denne forenklingen, og har derfor nærmest blitt et synonym til denne teknologien. Grunnen til at Ikomm nå setter sitt syn mot denne teknologien, er at Docker og Microsoft inngikk et samarbeid i slutten av 2014, og Docker sin teknologi kommer til å være implementert i Windows Server 2016.[100]

A.8 Prosjektorganisering

A.8.1 Ansvarsforhold og roller

Vår oppdragsgiver er IKOMM AS på Lillehammer, de er en sterk fagressurs innenfor dette feltet. De vil forsyne oss med den programvaren og maskinvaren vi trenger for å fullføre dette prosjektet på en god måte. Førsteamanuensis Erik Hjelmås ved NTNU Gjøvik er vår veileder. Han vil være vil til stor nytte for oss teoretisk og teknisk innenfor dette feltet, ettersom han er ansvarlig for kurs som har sterke relasjoner til dette emnet.

Andreas Dukstad er valgt ut som prosjektleder av resten av gruppa og vil ha ansvaret for å fordele arbeidsoppgaver mellom medlemmene i gruppa. Prosjektlederen vil lede an møtene våre og har ansvaret med å holde et overblikk i prosjektets fremgang.

Svein Edmund Varfjell er dokumentansvarlig har ansvaret for å formatere prosjektet-dokumentet i LaTeX. I tillegg til dette kommer ansvaret med å kjøre gruppa ved behov for reise.

Åsmund Helland Bu er vevmester og har ansvaret for å redigere prosjektets nettside. Ved en oppdatering er det vevmester som vil legge inn nytt og-/eller endre nettsidens innhold.

Jan Kristiansen vil ha ansvaret for å skrive referat på møter.

Alle i gruppa vil ta del i ansvaret med å løse teoretiske og tekniske utfordringer som vi kan møte i prosjektetsarbeid.

A.8.2 Rutiner, regler og statusmøter

Det blir holdt ukentlige statusmøter med veileder og møter oppdragsgiver etter avtale. Vi har flere faste ukentlige møter i gruppen der vi jobber og fordeler videre arbeidsoppgaver.

Tidspunktet for disse vil utvikle seg utover våren basert på når gruppemedlemmene har tid ut fra øvrige fag.

§1 Fravær fra faste møtetider skal rapporteres til prosjektleder (Andreas) ved første mulighet.
§2 Dersom et medlem ikke utfører arbeidsoppgaver skal veileder kontaktes dersom dette gjentar seg.
§3 Dersom gruppeleder og veileder er enig, kan medlemmer ekskluderes.
§4 Avgjørelser internt i gruppen avgjøres ved avstemming, dersom det oppstår en uenighet skal prosjektleder beslutte løsningen.
§5 Gruppeleder har rett til å signere på vegne av gruppen.
§6 Reise eller andre kostnader skal fordeles likt mellom alle gruppemedlemmene.

A.9 Planlegging, oppfølging og rapportering

Vi har valgt å bruke fossefallsmodellen i vårt prosjekt siden hver fase bygger på resultatet fra den forrige. Dette vil gi en god oversikt på hvilke beslutninger vi skal foreta i forhold til resultatet i fasen. Hver fase i fossefallsmodellen har en klar hensikt og tar for seg hvilke arbeidsoppgaver som skal utføres. Dette vil gi en god oversikt på hva som skal gjøres og hva som er blitt gjort.

Fremdriftsplan

Det første punktet på fremdriftsplanen er å få satt opp et arbeidsmiljø hos Ikomm slik at vi kan gjennomføre casene som blir utarbeidet i samarbeid med Ikomm. Casene er for øyeblikket ikke ferdig utarbeidet og blir derfor bare referert til som “Case 1” og “Case 2” i denne prosjektplanen. I forkant av hver case er det satt av to dager til forberedelse og innhentelse av nødvendig informasjon for å gjennomføre den aktuelle casen. Det er i tillegg satt av en uke for å løse eventuelle problemer som har oppstått eller oppgaver som ikke har blitt fullført tidligere.

Oppsett av arbeidsmiljø

Vi skal sette opp en server hos Ikomm i begynnelsen av Februar. Maskinvaren får vi låne av Ikomm. Det vil bli hostet en Windows Server 2016 i deres datasenter som vi kan benytte som arbeidsmiljø. Vi kan da koble oss opp med VPN og jobbe fra NTNU i Gjøvik.

A.10 Organisering av kvalitetssikring

A.10.1 Versjonskontroll

Vi bruker Google Docs til rapportskriving. Google Docs har versjonskontroll, endringshistorikk og mulighet for å gå tilbake til en tidligere versjon av dokumentet. Det blir herifra stegvis ført inn i LaTeX. Vi bruker Dropbox for lagring av LaTeX-dokumenter, samt andre nødvendige filer. Dropbox har også versjonshåndtering.

A.10.2 Risikoanalyse

Vi har valgt å lage en enkel risikoanalyse med verdiene 1-2-3-4, hvor 1 er det den minst sannsynlige/lavest konsekvens. Disse verdiene er satt utifra våre egne erfaringer og skjønn. Med sannsynlighet så menes vi sannsynligheten for at en hendelse inntreffer under prosjektet. Med konsekvens mener vi konsekvensen en gitt hendelse vil ha for full-

føringen av prosjektet. Produktet av konsekvens og sannsynlighet gir oss risikoverdien. Vi har satt øverste tillatte risikoverdi til å være 7.

Testmiljø blir ikke satt opp

Store deler av oppgaven vil bli gjort ved at vi gjennomfører ulike caser vi har kommet frem til i samarbeid med Ikomm. For at dette skal være mulig, trenger vi et testmiljø hvor vi kan gjennomføre cases med de nødvendige lisenser og programvare. Dersom dette ikke er klart innen rimelig tid, vil oppgaven bli langt vanskeligere å gjennomføre og sluttresultatet vil i større grad være basert på teori, da vi ikke får testet teknologien mot de aktuelle systemene hos Ikomm. Utifra dette, vurderes konsekvensen dersom dette inntreffer til å være 4, da vi mister store deler av grunnlaget for å gjennomføre oppgaven. I første møte med Ikomm ble det ikke avtalt dato for når miljøet skulle være klart, men at det skulle gjøres innen rimelig tid. Vi har derfor vurdert sannsynligheten for at miljøet ikke blir gjort klar innenfor vår skisserte tidsplan til å være 2. Konsekvensen er satt utifra dersom miljøet ikke blir satt opp i det hele. Dersom det blir satt opp, men forsinket, vil konsekvensen øke i forhold til hvor stor forsinkelsen er.

Sannsynlighet	Konsekvens	Risiko
2	4	8

Tiltak : Vi kan sette opp et testmiljø på en av våre egne datamaskiner. Dette vil redusere konsekvensen fra 4 til 3, risikoverdien ender da opp på 6, innenfor vår aksepterte risikogrense.

Gruppesamarbeidet bryter sammen

Bachelorgruppen er sammensatt av fire studenter som alle har ulike valgfag under gjennomføringen av bacheloroppgaven. Møtetidspunkter for gruppen kan derfor bli en utfordring. Dersom det oppstår konflikter som ikke blir løst opp innen rimelig tid, kan det ha negative effekter på fremgangen i oppgaven. Konsekvensen av at samarbeidet slutter å fungere vurderer vi til å være 4, da oppgaven er av en slik størrelse som ikke lar seg gjennomføre uten en gruppe. Sannsynligheten for at dette blir et tilfelle vurderer vi til å være 1, da dette er noe som går utover oss alle.

Sannsynlighet	Konsekvens	Risiko
1	4	4

Tiltak: Selv om vi mener sannsynligheten er lav, implementerer vi likevel en løsning i form av tydelige grupperegler.

Ikomm ikke har mulighet til å bistå ved uventede problemer

Ikomm har lovet oss å sette opp en server med nødvendige lisenser som vi kan gjennomføre deler av oppgaven vår på. Ikomm har mange baller i luften og vil ikke alltid være tilgjengelig for oss. I tillegg til dette så er containerteknologien noe som Ikomm ikke har stor kompetanse innad (derav oppgaven vår). Dersom det skulle oppstå et problem som Ikomm enten ikke har tid til å hjelpe med, eller ikke kan hjelpe med, vil det forskyve tidsplanen vår og redusere tiden vi har mulighet til å bruke på det øvrige arbeidet. Konsekvensen av at problemer oppstår som Ikomm har mulighet til å hjelpe oss med trenger

ikke å nødvendigvis å være av stor betydning, da vi har en høy kompetanse innad i gruppen samt veileder. Vi har derfor vurdert konsekvensen til å være 2. Sannsynligheten av at et tilfelle som nevnt ovenfor inntreffer, ser vi på som relativt stor, og har derfor satt sannsynligheten til å være 3.

Sannsynlighet	Konsekvens	Risiko
3	2	6

Tiltak: Kontakte brukersupport for Docker eller se på dokumentasjonen til Docker. Dette vil ikke gjelde alle problemer, men da risikoverdien er under vår tillatte grense, ser vi på dette som et tilstrekkelig tiltak.

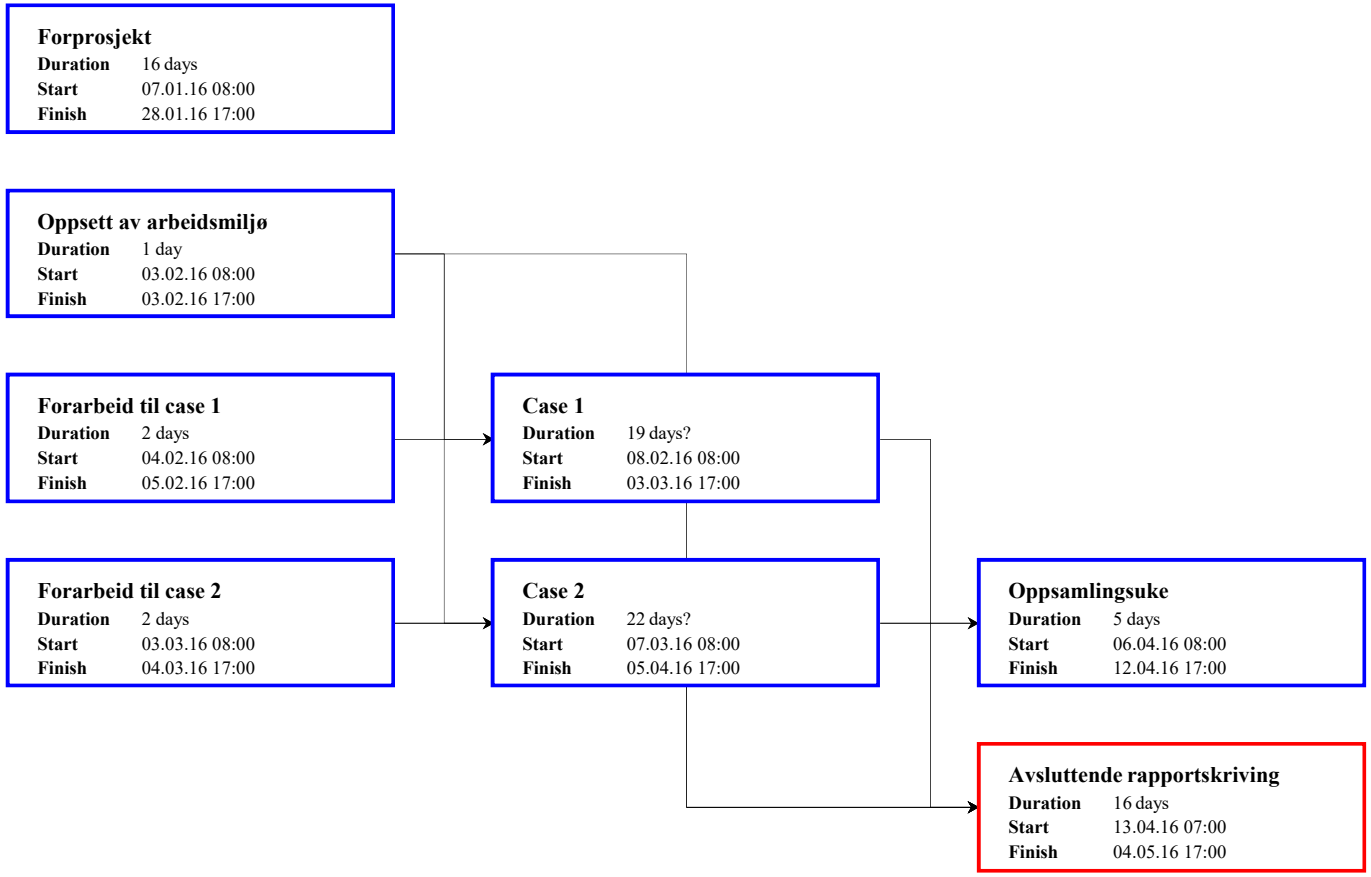
Tap av arbeid

Tap av arbeid og dets konsekvens avhenger veldig av hvor i prosessen vi befinner oss og omfanget av det som forsvinner. Det aller meste kan gjenskapes dersom tiden strekker til, det er kunnskapen som vi har tilegnet oss som er viktig. Konsekvensen av et tap vurderes derfor til å være 2, men 3 dersom det skjer mot slutten av oppgaven. Sannsynligheten for at vi taper data ser for øyeblikket lav ut, da vi bruker Google Docs som felles arbeidsområde, men alle har lokale kopier av denne. Sannsynligheten er derfor vurdert til å være 1.

Sannsynlighet	Konsekvens	Risiko
1	3	3

Tiltak: Hver mandag skal det tas ny lokal backup av dokumenter.

A.11 Plan for gjennomføring



Figur 61: Fremdriftsplan

TILLEGG

B

MØTELOGG OG TIMEFØRING

MØTELOGG	
14.12.2015 Gruppemøte Deltakere Nettbasert møte der gruppen planla videre arbeid over nyttår	Åsmund Bu Andreas Dukstad Jan Kristiansen Svein Edmund Varfjell
13.01.2016 Gruppemøte Deltakere Det ble gjort er rask innføring i containerteknologi og docker samt forberedelse til første møte med Ikomm og spørsmål som gruppen ønsket besvart. Det ble satt opp delt mappe i Google drive for bedre informasjonsflyt. Gruppen ble enig om å bruke LATEX til innføring av oppgaven.	Åsmund Bu Andreas Dukstad Jan Kristiansen Svein Edmund Varfjell

19.01.2016 Ukentlig møte med veileder Deltakere Gruppen fikk hjelp med å formulere spørsmål til oppdragsgiver. I tillegg ble det avtalt at det skulle være ukentlig møter med veileder tirsdager kl. 11.30.	Åsmund Bu Andreas Dukstad Jan Kristiansen Svein Edmund Varfjell Erik Hjelmås
20.01.2016 Innledende møte med oppdragsgiver Deltakere Oppgaven ble diskutert i detalj og det ble avtalt at Ikomm skulle finne to cases gruppen skulle gjennomføre som var relevant for deres drift. Casene var ikke utarbeidet enda og skulle komme i en e-post ved en senere anledning. Det ble avtalt at prosjektavtale skulle signeres og oversendes Ikomm. Ikomm skulle klargjøre en server som gruppen skulle hente ved en senere anledning.	Åsmund Bu Andreas Dukstad Jan Kristiansen Svein Edmund Varfjell Stig Syversen Dag Olav Nilsen
26.01.2016 Ukentlig møte med veileder Deltakere Tilbakemelding på prosjektplan, alle personlige pronomen skulle fjernes før innføring i LaTeX.	Åsmund Bu Andreas Dukstad Jan Kristiansen Svein Edmund Varfjell Erik Hjelmås
02.02.2016 Ukentlig møte med veileder Deltakere Veileder gav oss tilgang til en backlog med “Computer World” for å finne eksempler på bruk av Docker i andre bedrifter.	Åsmund Bu Andreas Dukstad Jan Kristiansen Svein Edmund Varfjell Erik Hjelmås

09.02.2016 Ukentlig møte med veileder Deltakere Det ble avtalt hvordan server skal settes opp i skolens nettverk. Det ble diskutert rundt case 1 og hvordan VDI-løsning kan gjennomføres.	Åsmund Bu Andreas Dukstad Jan Kristiansen Svein Edmund Varfjell Erik Hjelmås
16.02.2016 Ukentlig møte med veileder Deltakere Det ble diskutert rundt hvilke image gruppen bør basere seg på (nanoserver eller windows server core) og ettersom nanoserver er headless ble det bestemt at Windows server core skulle kjøres.	Åsmund Bu Andreas Dukstad Jan Kristiansen Svein Edmund Varfjell Erik Hjelmås
23.02.2016 Ukentlig møte med veileder Deltakere Gruppen fikk uttrykt problematikken med VDI-løsningen som tilsynelatende ikke fungerte i technical preview 4. Gruppen skal gi VDI et nytt forsøk når technical preview 5 kommer ut, men foreløpig var gruppen nødt til å begynne med demo til fremvisning for Ikomm. Denne skulle gå ut på oppsett av webservere i containere.	Åsmund Bu Andreas Dukstad Jan Kristiansen Svein Edmund Varfjell Erik Hjelmås
01.03.2016 Ukentlig møte med veileder Deltakere	Åsmund Bu Andreas Dukstad Jan Kristiansen Svein Edmund Varfjell Erik Hjelmås

Det ble diskutert hvordan webserverene i demoen skulle være unike og hvordan en kunne bruke portmapping for å få ulike webservere på ulike adresser ellers skulle tiden videre brukes på demo og på å fylle inn i rapporten.	
08.03.2016 Ukentlig møte med veileder Deltakere Demo ble forklart og delvis fremvist for veileder, gruppen ble bedt om å lage video av demoutførelse for å ha i bakhånd dersom noe skulle gå galt under demoen for Ikomm, Andreas ble valgt som ansvarlig for dette.	Åsmund Bu Andreas Dukstad Jan Kristiansen Svein Edmund Varfjell Erik Hjelmås
14.03.2016 Demoutførelse for oppdragsgiver Deltakere Demoutførelse for oppdragsgiver. Gruppen fikk problemer med brannmur underveis i demo, brukte video av demo for å fullføre fremvisningen. Gruppen forklarte oppdragsgiver hvordan en VDI-løsning vil være vanskelig i Docker, det ble diskutert hvilke andre cases de kunne trenge. Det ble ikke gjort en beslutning om hvilken ny case oppdragsgiver kunne tilby og denne skulle sendes til oss i etterkant.	Åsmund Bu Andreas Dukstad Jan Kristiansen Svein Edmund Varfjell Dag Olav Nilsen
15.03.2016 Ukentlig møte med veileder Deltakere Veileder ble informert om hvordan demoutførelsen hadde gått. Veileder ønsket at gruppen skulle videre utforske dynamisk skalering med webservere. I tillegg skulle gruppen finne gode bruksområder for containere og bruksområder hvor container ikke er å foretrekke.	Åsmund Bu Andreas Dukstad Jan Kristiansen Svein Edmund Varfjell Erik Hjelmås

05.04.2016 Ukentlig møte med veileder Deltakere Veileder ønsket at gruppen skulle fortsette med skalering i tillegg til å forsøke å finne eksempler på lastbalansering gjort i Linuxcontainere.	Andreas Dukstad Jan Kristiansen Erik Hjelmås
12.04.2016 Ukentlig møte med veileder Deltakere Grunnet avbrudd fra møter grunnet rapportskriving fikk veileder en situasjonsrapport på fremgang i rapporten. Det ble avtalt levering av utkast av rapport til veileder onsdag 11.Mai. Veileder ble i tillegg informert om Svein Edmund sin sykdomsperiode.	Åsmund Bu Andreas Dukstad Jan Kristiansen Erik Hjelmås
19.04.2016 Ukentlig møte med veileder Deltakere Veileder ønsket at gruppen skulle kontakte representant for Azure for å få bruke Azure til å oppsett av containere for å demonstrere dette, mail angående dette ble sendt under møtet.	Åsmund Bu Andreas Dukstad Jan Kristiansen Svein Edmund Varfjell Erik Hjelmås
26.04.2016 Ukentlig møte med veileder Deltakere Møte neste uke utgår grunnet at veileder er bortreist. Gruppen skal utføre ytelsestesting på containere mot virtuelle maskiner, det ble diskutert ulike verktøy for ytelsestesting da det er behov for et verktøy som kan kjøres uten GUI og direkte i kommandolinje.	Åsmund Bu Andreas Dukstad Jan Kristiansen Svein Edmund Varfjell Erik Hjelmås
10.05.2016 Ukentlig møte med veileder	

Deltakere	Åsmund Bu Andreas Dukstad Jan Kristiansen Svein Edmund Varfjell Erik Hjelmås
Tilbakemelding på utkast av rapport. Veileder ønsket mer utforsking av OpenStack Magnum, feilretting på figurtekster samt stedvis bedre setningsoppbygning. I tillegg skal I/O-testing utforskes videre.	

Tabell 5: Møtelogg

TIMEFØRING

NÅR	VARIGHET	SPESIELLE MERKNADER
07.01.2016	4	Lesing om containerteknologi
13.01.2016	8	Innføring i containere, planlegging av første møte med Ikomm
14.01.2016	6	Lesing og notering om containere/docker etc.
14.01.2016	8	Lesing om containere og docker etc.
19.01.2016	8	Ukentlig møte med veileder + research
20.01.2016	8	Møte med IKOMM samt arbeid med prosjektplan
21.01.2016	8	Arbeid med prosjektplan
23.01.2016	3	Lesing/prosjektplan
23.01.2016	4	Oppretting av LaTeX document og lest det som er skrevet så langt
25.01.2016	8	Arbeid med prosjektplan
26.01.2016	8	Arbeid med prosjektplan + Tilbakemelding på utkast av plan
26.01.2016	3	LaTeX og rettskriving
27.01.2016	8	Arbeid med prosjektplan
28.01.2016	10	Avsluttende arbeid med prosjektplan
01.02.2016	2	Sett opptak av Kyrres forelesning om docker
02.02.2016	3	Møte med veileder og lesing av Computerworld
03.02.2016	8	Henting og oppsett av Server
04.02.2016	6	
08.02.2016	3	Sette opp server på serverrom på skolen
09.02.2016	2	Satt opp server i nettverket
09.02.2016	8	Research

10.02.2016	10	Feilsøking, reinstallering av nyere versjon av Server 2016
11.02.2016	8	
15.02.2016	9	Feilsøking og oppretting av container hosts(nano og core server)
16.02.2016	8	Ukentlig møte med veileder + arbeid med containerhost
17.02.2016	8	Fortsettelse testing av container og skrive om containere
18.02.2016	8	
22.02.2016	8	Oppsett av VDI, certificate,
23.02.2016	4	Møte med veileder og gjenoppretting av server
24.02.2016	6	Gjenoppretting av server
25.02.2016	8	Start med demo for ikomm
29.02.2016	14	Preparasjon av demo
01.03.2016	4	Møte med veilleder og demo
02.03.2016	8	dynamisk skalering og demo
03.03.2016	8	Videre arbeid med demo
07.03.2016	8	Arbeid med demo, grunnarbeid med dynamisk skalering
08.03.2016	7	Ukentlig møte med veileder og dynamisk skalering
09.03.2016	8	Avsluttende arbeid med demo
10.03.2016	6	
11.03.2016	6	Laget video av demogjennomføring for å ha i bakhånd om noe går galt
14.03.2016	4	Tur til Ikomm for demoutførelse
29.03.2016-13.04.2016		Svein Edmund Syk
15.03.2015	8	Møte med veilleder samt dynamisk skalering
16.03.2016	8	Arbeid med dynamisk skalering + rapport
17.03.2016	8	Dynamisk skalering + rapport
28.03.2016	8	Sikkerhetsaspekter med docker + rapport
29.03.2016	8	
30.03.2016	8	
31.03.2016	8	
04.04.2016	10	
06.04.2016	8	
07.04.2016	8	
11.04.2016	8	
12.04.2016	8	
13.04.2016	8	
14.04.2016	8	
18.04.2016	10	Research av docker swarm
19.04.2016	8	
20.04.2016	8	
21.04.2016	10	Forbedring av demo for rapport
25.04.2016	8	Ført inn arbeid i Latex

26.04.2016	8	Ytelsestesting
27.04.2016	9	Ytelsestesting
28.04.2016	8	Ytelsestesting
02.05.2016	10	Ytelsestesting
03.05.2016	12	Latex + Ytelsestesting
04.05.2016	12	Latex + Ytelsestesting
05.05.2016	12	Latex + Ytelsestesting
06.05.2016	10	Ytelsestesting, forberedelse av utkast
07.05.2016	10	Ytelsestesting, forberedelse av utkast
08.05.2016	12	Forberedelse av utkast, start-tider på containere og vm
09.05.2016	12	Forberedelse av utkast
10.05.2016	12	Endringer etter tilbakemeldinger på utkast
11.05.2016	10	Endringer etter tilbakemeldinger på utkast
12.05.2016	10	Endringer etter utkast + Demonstrasjon skjuling av pid
13.05.2016	10	Endringer etter utkast + I/O-test
14.05.2016	10	I/O-test + nettverksytelse
15.05.2016	10	Ferdigstilling av tekst til rapport
16.05.2016	10	Finpuss av rapport
17.05.2016	5	Siste finpuss
Total	626	

Tabell 6: Timeføring

TILLEGG

C

PROSJEKTAVTALE

NTNU
Norges Teknisk-Naturvitenskapelige Universitet
NTNU i Gjøvik, Avd. Informatikk og Medieteknikk

PROSJEKTAVTALE

mellom NTNU v/Avd. Informatikk og Medieteknikk (NTNU/AIMT) (utdanningsinstitusjon), og

Ikomm AS (oppdragsgiver), og Andreas Dukstad, Jan Kristiansen, Svein Edmund Varfjell, Åsmund Helland Bu (student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 01.01.2016 til 18.05.2016.

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der AIMT yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra AIMT å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon/fax, reiser og nødvendig overnatting på steder langt fra Gjøvik/AIMT. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. AIMT står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må ansees som en eksamensrelatert oppgave som blir bedømt av faglærer/ veileder og sensor (intern og ekstern sensor). Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv hvis de har skriftlig karakter A, B eller C.

   Tilgjengeliggjøring i det åpen arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og dekan om de i løpet av prosjektet endrer syn på slik offentliggjøring.

   Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av AIMT til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU/AIMT og/eller studenter har interesser.

6. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

7. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/ veileder stiller til disposisjon.

8. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i Fronter. I tillegg leveres et eksemplar til oppdragsgiver.

9. Denne avtalen utferdiges med et eksemplar til hver av partene. På vegne av AIMT er det dekan/prodekan som godkjenner avtalen.

10. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og AIMT som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten AIMT som partner.

11. Når NTNU/AIMT også opptrer som oppdragsgiver, trer NTNU/AIMT inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

12. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene i mellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

13. Deltakende personer ved prosjektgjennomføringen:

NTNU/AIMTs veileder (navn): Erik Iljelmås

Oppdragsgivers kontaktperson (navn): _____

Student(er) (signatur): Jan Kristiansen dato 17.02.16
Asmund Helland BU dato 17.02.16
Sveinz. Varfjell dato 17.02.16
Andreas Dukstad dato 17.02.16

Oppdragsgiver (signatur): _____ dato 17.02.16

Signert avtale leveres digitalt i Fronter(IMT3912)
Godkjennes digitalt av AIMTs dekan

Om papirversjon med signatur er ønskelig, må papirversjon leveres til AIMT i tillegg.
Plass for evt sign:
AIMT Dekan/prodekan (signatur): _____ dato _____

TILLEGG

D

SCRIPT

```
############################################################
# Script assembled with makeps1.js from
# New-ContainerHost-Source.ps1
# ..\common\ContainerHost-Common.ps1
# Unattend-Source.ps1
# New-ContainerHost-Main.ps1
############################################################

<#
    .NOTES
        Copyright (c) Microsoft Corporation.  All rights reserved.

    .SYNOPSIS
        Create a VM as a new container host

    .DESCRIPTION
        Collects collateral required to create a container host
        Creates a VM
        Configures the VM as a new container host

    .PARAMETER DockerPath
        Path to a private Docker.exe.  Defaults to https://aka.ms/tp4/docker

    .PARAMETER Password
        Password for the built-in Administrator account.

    .PARAMETER HyperV
        If passed, prepare the machine for Hyper-V containers

    .PARAMETER NatSubnetPrefix
        Prefix for container hosts NAT range.

    .PARAMETER ScriptPath
        Path to a private Install-ContainerHost.ps1.  Defaults to
        https://aka.ms/SetupContainers

    .PARAMETER SkipDocker
        If passed, skips Docker install

    .PARAMETER SwitchName
        Specify a switch to give the container host network connectivity

    .PARAMETER UnattendPath
        Path to custom unattend.xml for use in the container host VM.  If not passed, a
        default
        unattend.xml will be used that contains a built-in Administrator account

    .PARAMETER VhdPath
        Path to a private Windows Server image.

    .PARAMETER VmName
        Friendly name for container host VM to be created.  Required.

    .PARAMETER WimPath
        Path to a private .wim file that contains the base package image.  Only required if
        -VhdPath is also passed
```

```
    .PARAMETER WindowsImage
        Image to use for the VM.  One of NanoServer, ServerDatacenter, or
        ServerDatacenterCore [default]

    .EXAMPLE
        .\Install-ContainerHost.ps1 -SkipDocker

#>
#Requires -Version 4.0

[CmdletBinding(DefaultParameterSetName="IncludeDocker")]
param(
    [Parameter(ParameterSetName="IncludeDocker")]
    [string]
    [ValidateNotNullOrEmpty()]
    $DockerPath = "https://aka.ms/tp4/docker",

    [switch]
    $HyperV,

    [Parameter(ParameterSetName="IncludeDocker")]
    [Parameter(ParameterSetName="SkipDocker")]
    [string]
    [ValidateNotNullOrEmpty()]
    $IsoPath = "https://aka.ms/tp4/serveriso",

    [string]
    $NATSubnetPrefix = "172.16.0.0/24",

    [Parameter(ParameterSetName="IncludeDocker", Mandatory, Position=1)]
    [Parameter(ParameterSetName="SkipDocker", Mandatory, Position=1)]
    [Security.SecureString]
    $Password = ("P@ssw0rd" | ConvertTo-SecureString -AsPlainText -Force),

    [Parameter(ParameterSetName="Prompt", Mandatory)]
    [switch]
    $Prompt,

    [string]
    [ValidateNotNullOrEmpty()]
    $ScriptPath = "https://aka.ms/tp4/Install-ContainerHost",

    [Parameter(ParameterSetName="SkipDocker", Mandatory)]
    [switch]
    $SkipDocker,

    [Parameter(ParameterSetName="Staging", Mandatory)]
    [switch]
    $Staging,

    [string]
    $SwitchName,

    [string]
    [ValidateNotNullOrEmpty()]
    $UnattendPath,
```

```
    [string]
    [ValidateNotNullOrEmpty()]
    $VhdPath,

    [Parameter(ParameterSetName="IncludeDocker", Mandatory, Position=0)]
    [Parameter(ParameterSetName="SkipDocker", Mandatory, Position=0)]
    [Parameter(ParameterSetName="Staging", Mandatory, Position=0)]
    [string]
    [ValidateNotNullOrEmpty()]
    $VmName,

    [string]
    [ValidateNotNullOrEmpty()]
    $WimPath,

    [string]
    [ValidateSet("NanoServer", "ServerDatacenter", "ServerDatacenterCore")]
    $WindowsImage = "ServerDatacenterCore"
)

if ($Prompt)
{
    if ((Get-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V).State -ne
    "Enabled")
    {
        throw "Hyper-V must be enabled to continue"
    }

    $VmName = Read-Host 'Please specify a name for your VM'

    #
    # Do we require nesting?
    #
    $nestedChoiceList = New-Object
    System.Collections.ObjectModel.Collection[System.Management.Automation.Host.ChoiceDescript
    ion]

    $nestedChoiceList.Add((New-Object "System.Management.Automation.Host.ChoiceDescription"
    -ArgumentList "&No"))
    $nestedChoiceList.Add((New-Object "System.Management.Automation.Host.ChoiceDescription"
    -ArgumentList "&Yes"))

    $HyperV = [boolean]$Host.ui.PromptForChoice($null, "Would you like to enable Hyper-V
    containers?", $nestedChoiceList, 0)

    #
    # Which image?
    #
    $imageChoiceList = New-Object
    System.Collections.ObjectModel.Collection[System.Management.Automation.Host.ChoiceDescript
    ion]

    $imageChoiceList.Add((New-Object "System.Management.Automation.Host.ChoiceDescription"
    -ArgumentList "&NanoServer"))
    $imageChoiceList.Add((New-Object "System.Management.Automation.Host.ChoiceDescription"
    -ArgumentList "&ServerDatacenter"))
```

```
    $imageChoiceList.Add((New-Object "System.Management.Automation.Host.ChoiceDescription"
    -ArgumentList "ServerDatacenter&Core"))

    $imageIndex = $Host.ui.PromptForChoice($null, "Select your container host image",
    $imageChoiceList, 2)

    switch ($imageIndex)
    {
        0 {$WindowsImage = "NanoServer"}
        1 {$WindowsImage = "ServerDatacenter"}
        2 {$WindowsImage = "ServerDatacenterCore"}
    }

    #
    # Administrator password?
    #
    $Password = Read-Host 'Please specify Administrator password' -AsSecureString

    #
    # Install docker?
    #
    $dockerChoiceList = New-Object
    System.Collections.ObjectModel.Collection[System.Management.Automation.Host.ChoiceDescript
    ion]

    $dockerChoiceList.Add((New-Object "System.Management.Automation.Host.ChoiceDescription"
    -ArgumentList "&Yes"))
    $dockerChoiceList.Add((New-Object "System.Management.Automation.Host.ChoiceDescription"
    -ArgumentList "&No"))

    $SkipDocker = [boolean]$Host.ui.PromptForChoice($null, "Would you like to install
    Docker?", $dockerChoiceList, 0)


    if ($SkipDocker)
    {
        $global:ParameterSet = "SkipDocker"
    }
    else
    {
        $global:ParameterSet = "IncludeDocker"
    }
}
else
{
    $global:ParameterSet = $PSCmdlet.ParameterSetName
}

$global:WimSaveMode = $true
$global:PowerShellDirectMode = $true

#
# Image information
#
if ($WindowsImage -eq "NanoServer")
{
    $global:imageName = "NanoServer"
```

```
}
else
{
    $global:imageName = "WindowsServerCore"
}
$global:imageVersion = "10586.0"

#
# Branding strings
#
$global:brand = $WindowsImage
$global:imageBrand = "$($global:brand)_en-us_TP4_Container"
$global:isoBrandName = "$global:brand ISO"
$global:vhdBrandName = "$global:brand VHD"

#
# Get the management service settings
#
$global:localVhdRoot = "$((Get-VMHost).VirtualHardDiskPath)".TrimEnd("\")
$global:freeSpaceGB = 0

#
# Define a default VHD name if not specified
#
if ($VhdPath -and ($(Split-Path -Leaf $VhdPath) -match ".*\.vhdx?"))
{
    $global:localVhdName = $(Split-Path -Leaf $VhdPath)

    #
    # Assume this is an official Windows build VHD/X.  We parse the build number and QFE
    from the filename
    #
    if ($VhdPath -match "(\d{5})\.(\d{1,5}).*\.(vhdx)?")
    {
        $global:imageVersion = "$($Matches[1]).$($Matches[2])"
    }

    #
    # Save-ContainerImage doesn't work for internal shares yet
    #
    $global:WimSaveMode = $false
}
else
{
    $global:localVhdName = "$($global:imageBrand).vhd"
}

$global:localIsoName = "WindowsServerTP4.iso"
$global:localIsoPath = "$global:localVhdRoot\$global:localIsoName"
$global:localIsoVersion =
"$global:localVhdRoot\ContainerISOVersion.$($global:imageVersion).txt"

$global:localVhdPath = "$global:localVhdRoot\$global:localVhdName"
$global:localVhdVersion =
"$global:localVhdRoot\ContainerVHDVersion.$($global:imageVersion).txt"

$global:localWimName = "$global:imageName.wim"
```

```
$global:localWimVhdPath = "$global:localVhdRoot\$($global:imageName)-WIM.vhdx"
$global:localWimVhdVersion =
"$global:localVhdRoot\$($global:imageName)Version.$($global:imageVersion).txt"


function
Cache-HostFiles
{
    if ($(Test-Path $global:localVhdPath) -and
        $(Test-Path $global:localVhdVersion))
    {
        Write-Output "The latest $global:vhdBrandName is already present on this system."
    }
    else
    {
        if ($global:freeSpaceGB -le 20)
        {
            Write-Warning "You currently have only $global:freeSpaceGB GB of free space
            available; 20GB is required"
        }

        if ($(Test-Path $global:localVhdPath) -and
            -not (Test-Path $global:localVhdVersion))
        {
            Write-Warning "There is a newer $global:vhdBrandName available."
        }

        if ($VhdPath)
        {
            Write-Output "Copying $global:vhdBrandName from $VhdPath to
            $global:localVhdPath..."
            Copy-File -SourcePath $VhdPath -DestinationPath $global:localVhdPath
        }
        else
        {
            if (Test-Path $global:localIsoPath)
            {
                Write-Output "The latest $global:isoBrandName is already present on this
                system."
            }
            else
            {
                Write-Output "Copying $global:isoBrandName from $IsoPath to
                $global:localIsoPath (this may take several minutes)..."
                Copy-File -SourcePath $IsoPath -DestinationPath $global:localIsoPath
            }

            try
            {
                $convertScript = $(Join-Path $global:localVhdRoot "Convert-WindowsImage.ps1")

                Write-Verbose "Copying Convert-WindowsImage..."
                Copy-File -SourcePath 'https://aka.ms/tp4/Convert-WindowsImage'
                -DestinationPath $convertScript

                #
                # Dot-source until this is a module
```

```
            #
            . $convertScript

            Write-Output "Mounting ISO..."
            $openIso = Mount-DiskImage $global:localIsoPath

            # Refresh the DiskImage object so we can get the real information about it.
            I assume this is a bug.
            $openIso = Get-DiskImage -ImagePath $global:localIsoPath
            $driveLetter = ($openIso | Get-Volume).DriveLetter

            Write-Output "Converting WIM to VHD..."
            if ($WindowsImage -eq "NanoServer")
            {
                #
                # Workaround an issue in the RTM version of Convert-WindowsImage.ps1
                #
                if (Get-Module Hyper-V)
                {
                    Add-WindowsImageTypes
                }

                Import-Module "$($driveLetter):\NanoServer\NanoServerImageGenerator.psm1"

                if ($Staging)
                {
                    New-NanoServerImage -MediaPath "$($driveLetter):\" -TargetPath
                    $global:localVhdPath -Containers -ReverseForwarders -GuestDrivers
                    -AdministratorPassword $Password
                }
                else
                {
                    New-NanoServerImage -MediaPath "$($driveLetter):\" -TargetPath
                    $global:localVhdPath -Compute -Containers -ReverseForwarders
                    -GuestDrivers -AdministratorPassword $Password
                }
            }
            else
            {
                Convert-WindowsImage -DiskLayout BIOS -SourcePath
                "$($driveLetter):\sources\install.wim" -Edition $WindowsImage -VhdPath
                $global:localVhdPath
            }
        }
        catch
        {
            throw $_
        }
        finally
        {
            Write-Output "Dismounting ISO..."
            Dismount-DiskImage $global:localIsoPath
        }
    }

    "This file indicates the web version of the base VHD" | Out-File -FilePath
    $global:localVhdVersion
```

```
    }

    if ($global:WimSaveMode -or $WimPath)
    {
        #
        # The combo VHD already contains the WIM.  Only cache if we are NOT using the combo
        VHD.
        #
        if ($(Test-Path $global:localWimVhdPath) -and
            $(Test-Path $global:localWimVhdVersion))
        {
            Write-Output "$global:brand Container OS Image (WIM) is already present on this
            system."
        }
        else
        {
            if ($(Test-Path $global:localWimVhdPath) -and
                -not (Test-Path $global:localWimVhdVersion))
            {
                Write-Warning "Wrong version of Container OS Image (WIM) inside
                $global:localWimVhdPath..."
                Remove-Item $global:localWimVhdPath
            }

            Write-Output "Creating temporary VHDX for the Containers OS Image WIM..."
            $dataVhdx = New-VHD -Path $global:localWimVhdPath -Dynamic -SizeBytes 8GB
            -BlockSizeBytes 1MB

            $disk = $dataVhdx | Mount-VHD -PassThru

            try
            {
                Write-Output "Initializing disk..."
                Initialize-Disk -Number $disk.Number -PartitionStyle MBR

                #
                # Create single partition
                #
                Write-Verbose "Creating single partition..."
                $partition = New-Partition -DiskNumber $disk.Number -Size
                $disk.LargestFreeExtent -MbrType IFS -IsActive

                Write-Verbose "Formatting volume..."
                $volume = Format-Volume -Partition $partition -FileSystem NTFS -Force
                -Confirm:$false

                $partition | Add-PartitionAccessPath -AssignDriveLetter

                $driveLetter = (Get-Volume |? UniqueId -eq $volume.UniqueId).DriveLetter

                if ($WimPath)
                {
                    Write-Output "Saving private Container OS image ($global:imageName)
                    (this may take a few minutes)..."
                    Copy-File -SourcePath $WimPath -DestinationPath
                    "$($driveLetter):\$global:localWimName"
                }
```

```
                else
                {
                    $imageVersion = "10.0.$global:imageVersion"
                    Write-Output "Saving Container OS image ($global:imageName) version
                    $imageVersion from OneGet to $($driveLetter): (this may take a few
                    minutes)..."
                    Test-ContainerProvider

                    #
                    # TODO: should be error action stop by default
                    #
                    Save-ContainerImage $global:imageName -Version $imageVersion
                    -Destination "$($driveLetter):\$global:localWimName" -ErrorAction Stop
                }

                if (-not (Test-Path "$($driveLetter):\$global:localWimName"))
                {
                    throw "Container image not saved successfully"
                }

                "This file indicates the web version of the image WIM VHD" | Out-File
                -FilePath $global:localWimVhdVersion
            }
            catch
            {
                throw $_
            }
            finally
            {
                Write-Output "Dismounting VHD..."
                Dismount-VHD -Path $dataVhdx.Path
            }
        }
    }
}


function
Add-Unattend
{
    [CmdletBinding()]
    param(
        [string]
        $DriveLetter
    )

    $unattendFilePath = "$($DriveLetter):\unattend.xml"

    if ($UnattendPath -ne "")
    {
        Copy-File -SourcePath $UnattendPath -DestinationPath $unattendFilePath

        $unattendFile = New-Object XML
        $unattendFile.Load($unattendFilePath)

        Write-Output "Writing custom unattend.xml..."
    }
```

```
    else
    {
        $credential = New-Object System.Management.Automation.PsCredential("Administrator",
        $Password)

        $unattendFile = (Get-Unattend -Password
        $credential.GetNetworkCredential().Password).Clone()

        Write-Output "Writing default unattend.xml..."
    }

    if (-not $global:PowerShellDirectMode)
    {
        Write-Output "Configuring Install-ContainerHost.ps1 to run at first launch..."

        $installCommand = "%SystemDrive%\Install-ContainerHost.ps1 "

        if ($SkipDocker)
        {
            $installCommand += '-SkipDocker '
        }
        elseif ($Staging)
        {
            $installCommand += '-Staging '
        }
        else
        {
            $installCommand += '-DockerPath %SystemRoot%\System32\docker.exe '
        }

        if ($global:WimSaveMode)
        {
            $installCommand += "-WimPath 'D:\$global:localWimName' "
        }

        try
        {
            [System.Xml.XmlNamespaceManager]$nsmgr = $unattendFile.NameTable

            $nsmgr.AddNamespace('urn', "urn:schemas-microsoft-com:unattend")
            $nsmgr.AddNamespace('wcm', "http://schemas.microsoft.com/WMIConfig/2002/State")

            $firstLogonElement = $unattendFile.CreateElement("FirstLogonCommands",
            $nsmgr.LookupNamespace("urn"))

            $synchronousCommandElement = $unattendFile.CreateElement("SynchronousCommand",
            $nsmgr.LookupNamespace("urn"))
            $synchronousCommandElement.SetAttribute("action", $nsmgr.LookupNamespace("wcm"),
            "add") | Out-Null

            $commandLineElement = $unattendFile.CreateElement("CommandLine",
            $nsmgr.LookupNamespace("urn"))
            $commandLineElement.InnerText =
            "%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell -NoLogo -NonInteractive
            -ExecutionPolicy Unrestricted -Command ""& { $installCommand } "" "

            $descriptionElement = $unattendFile.CreateElement("Description",
```

```
            $nsmgr.LookupNamespace("urn"))
            $descriptionElement.InnerText = "Running Containers Host setup script"

            $orderElement = $unattendFile.CreateElement("Order",
            $nsmgr.LookupNamespace("urn"))
            $orderElement.InnerText = "1"

            $synchronousCommandElement.AppendChild($commandLineElement) | Out-Null
            $synchronousCommandElement.AppendChild($descriptionElement) | Out-Null
            $synchronousCommandElement.AppendChild($orderElement) | Out-Null

            $firstLogonElement.AppendChild($synchronousCommandElement) | Out-Null

            $oobeSystemNode = $unattendFile.unattend.settings |? pass -eq "oobeSystem"
            $shellSetupNode = $oobeSystemNode.component |? name -eq
            "Microsoft-Windows-Shell-Setup"

            $shellSetupNode.AppendChild($firstLogonElement) | Out-Null
        }
        catch
        {
            Write-Warning "Failed to modify unattend.xml.  Please manually run 'powershell
            $installCommand' in the VM"
        }

    }

    $unattendFile.Save($unattendFilePath)
}


function
Edit-BootVhd
{
    [CmdletBinding()]
    param(
        [string]
        $BootVhdPath,

        [bool]
        $IncludeDocker
    )

    #
    # Protect this with a mutex
    #
    $mutex = New-Object System.Threading.Mutex($False, $global:imageName);

    $bootVhd = Get-Vhd $BootVhdPath

    try
    {
        Write-Output "VHD mount must be synchronized with other running instances of this
        script.  Waiting for exclusive access..."
        $mutex.WaitOne() | Out-Null;
        Write-Verbose "Mutex acquired."
```

```
        Write-Output "Mounting $global:vhdBrandName for offline processing..."
        $disk = $bootVhd | Mount-VHD -PassThru | Get-Disk

        #
        # We can assume there is one partition/volume
        #
        $driveLetter = ($disk | Get-Partition | Get-Volume).DriveLetter

        if ($WindowsImage -eq "NanoServer")
        {
            if ((Test-Path $global:localIsoPath) -and $Staging)
            {
                #
                # Add packages
                #
                try
                {
                    Write-Output "Mounting ISO..."
                    $openIso = Mount-DiskImage $global:localIsoPath

                    # Refresh the DiskImage object so we can get the real information about
                    it.  I assume this is a bug.
                    $openIso = Get-DiskImage -ImagePath $global:localIsoPath
                    $isoDriveLetter = ($openIso | Get-Volume).DriveLetter

                    #
                    # Copy all packages into the image to make it easier to add them later
                    (at the cost of disk space)
                    #
                    Write-Output "Copying Nano packages into image..."
                    Copy-Item "$($isoDriveLetter):\NanoServer\Packages"
                    "$($driveLetter):\Packages" -Recurse
                }
                catch
                {
                    throw $_
                }
                finally
                {
                    Write-Output "Dismounting ISO..."
                    Dismount-DiskImage $global:localIsoPath
                }
            }
        }
        else
        {
            if ($global:PowerShellDirectMode)
            {
                #
                # Enable containers feature.  This saves a reboot
                #
                Write-Output "Enabling Containers feature on drive $driveLetter..."
                Enable-WindowsOptionalFeature -FeatureName Containers -Path
                "$($driveLetter):"  | Out-Null

                if ($HyperV)
                {
```

```
                    Write-Output "Enabling Hyper-V feature on drive $driveLetter..."
                    Enable-WindowsOptionalFeature -FeatureName Microsoft-Hyper-V -Path
                    "$($driveLetter):"  | Out-Null
                }
            }
            else
            {
                # Windows 8.1 DISM cannot operate on Windows 10 guests.
            }
        }

        if ($IncludeDocker)
        {
            #
            # Copy docker
            #
            Write-Output "Copying Docker into $global:vhdBrandName..."
            Copy-File -SourcePath $DockerPath -DestinationPath
            "$($driveLetter):\Windows\System32\docker.exe"

            if ($WindowsImage -ne "NanoServer")
            {
                Write-Output "Copying NSSM into $global:vhdBrandName..."
                Get-Nsmm -Destination "$($driveLetter):\Windows\System32"
            }
        }

        #
        # Add unattend
        #
        Add-Unattend $driveLetter

        #
        # Add Install-ContainerHost.ps1
        #
        Write-Output "Copying Install-ContainerHost.ps1 into $global:vhdBrandName..."
        Copy-File -SourcePath $ScriptPath -DestinationPath
        "$($driveLetter):\Install-ContainerHost.ps1"
    }
    catch
    {
        throw $_
    }
    finally
    {
        Write-Output "Dismounting VHD..."
        Dismount-VHD -Path $bootVhd.Path

        $mutex.ReleaseMutex()
    }
}


function
New-ContainerHost()
{
    Write-Output "Using VHD path $global:localVhdRoot"
```

```
try
{
    $global:freeSpaceGB = [float]((Get-Volume -DriveLetter
    $global:localVhdRoot[0]).SizeRemaining / 1GB)
}
catch
{
    Write-Warning "Cannot detect volume free space at $global:localVhdRoot"
}

#
# Validate network configuration
#
if ($SwitchName -eq "")
{
    $switches = (Get-VMSwitch |? SwitchType -eq "External")

    if ($switches.Count -gt 0)
    {
        $SwitchName = $switches[0].Name
    }
}

if ($SwitchName -ne "")
{
    Write-Output "Using external switch $SwitchName"
}
elseif ($Staging)
{
    Write-Output "No external virtual switch connectivity; OK for staging mode"
}
else
{
    throw "This script requires an external virtual switch.  Please configure a virtual
    switch (New-VMSwitch) and re-run."
}

#
# Get prerequisites
#
Cache-HostFiles

if ($(Get-VM $VmName -ea SilentlyContinue) -ne $null)
{
    throw "VM name $VmName already exists on this host"
}

#
# Prepare VHDs
#
Write-Output "Creating VHD files for VM $VmName..."

if ($global:WimSaveMode)
{
    $wimVhdPath = "$global:localVhdRoot\$VmName-WIM.vhdx"
    if (Test-Path $wimVhdPath)
    {
```

```
            Remove-Item $wimVhdPath
        }

        $wimVhd = New-VHD -Path "$wimVhdPath" -ParentPath $global:localWimVhdPath
        -Differencing -BlockSizeBytes 1MB
    }

    if ($global:freeSpaceGB -le 10)
    {
        Write-Warning "You currently have only $global:freeSpaceGB GB of free space
        available at $global:localVhdRoot)"
    }

    $global:localVhdPath -match "\.vhdx?" | Out-Null

    $bootVhdPath = "$global:localVhdRoot\$($VmName)$($matches[0])"
    if (Test-Path $bootVhdPath)
    {
        Remove-Item $bootVhdPath
    }
    $bootVhd = New-VHD -Path "$bootVhdPath" -ParentPath $global:localVhdPath -Differencing

    Edit-BootVhd -BootVhdPath $bootVhdPath -IncludeDocker $($global:ParameterSet -eq
    "IncludeDocker")

    #
    # Create VM
    #
    Write-Output "Creating VM $VmName..."
    $vm = New-VM -Name $VmName -VHDPath $bootVhd.Path -Generation 1

    Write-Output "Configuring VM $($vm.Name)..."
    $vm | Set-VMProcessor -Count ([Math]::min((Get-VMHost).LogicalProcessorCount, 64))
    $vm | Get-VMDvdDrive | Remove-VMDvdDrive
    $vm | Set-VM -DynamicMemory | Out-Null

    if ($SwitchName -eq "")
    {
        $switches = (Get-VMSwitch |? SwitchType -eq "External")

        if ($switches.Count -gt 0)
        {
            $SwitchName = $switches[0].Name
        }
    }

    if ($SwitchName -ne "")
    {
        Write-Output "Connecting VM to switch $SwitchName"
        $vm | Get-VMNetworkAdapter | Connect-VMNetworkAdapter -SwitchName "$SwitchName"
    }

    if ($HyperV)
    {
        #
        # Enable nested to support Hyper-V containers
        #
```

```
        $vm | Set-VMProcessor -ExposeVirtualizationExtensions $true

        #
        # Disable dynamic memory
        #
        $vm | Set-VMMemory -DynamicMemoryEnabled $false
    }

    if ($global:WimSaveMode)
    {
        #
        # Add WIM VHD
        #
        $wimHardDiskDrive = $vm | Add-VMHardDiskDrive -Path $wimVhd.Path -ControllerType SCSI
    }

    if ($Staging -and ($WindowsImage -eq "NanoServer"))
    {
        Write-Output "NanoServer VM is staged..."
    }
    else
    {
        Write-Output "Starting VM $($vm.Name)..."
        $vm | Start-VM | Out-Null

        Write-Output "Waiting for VM $($vm.Name) to boot..."
        do
        {
            Start-Sleep -sec 1
        }
        until (($vm | Get-VMIntegrationService |? Id -match
        "84EAAE65-2F2E-45F5-9BB5-0E857DC8EB47").PrimaryStatusDescription -eq "OK")

        Write-Output "Connected to VM $($vm.Name) Heartbeat IC."

        if ($global:PowerShellDirectMode)
        {
            $credential = New-Object
            System.Management.Automation.PsCredential("Administrator", $Password)

            $psReady = $false

            Write-Output "Waiting for specialization to complete (this may take a few
            minutes)..."
            $startTime = Get-Date

            do
            {
                $timeElapsed = $(Get-Date) - $startTime

                if ($($timeElapsed).TotalMinutes -ge 30)
                {
                    throw "Could not connect to PS Direct after 30 minutes"
                }

                Start-Sleep -sec 1
                $psReady = Invoke-Command -VMName $($vm.Name) -Credential $credential
```

```
        -ScriptBlock { $True } -ErrorAction SilentlyContinue
    }
    until ($psReady)

    Write-Verbose "PowerShell Direct connected."

    $guestScriptBlock =
    {
        [CmdletBinding()]
        param(
            [Parameter(Position=0)]
            [string]
            $WimName,

            [Parameter(Position=1)]
            [string]
            $ParameterSetName,

            [Parameter(Position=2)]
            [bool]
            $HyperV,

            [Parameter(Position=3)]
            [string]
            $NATSubnetPrefix
            )

        Write-Verbose "Onlining disks..."
        Get-Disk | ? IsOffline | Set-Disk -IsOffline:$false

        Write-Output "Completing container install..."
        $installCommand = "$($env:SystemDrive)\Install-ContainerHost.ps1 -PSDirect
        -NATSubnetPrefix $NATSubnetPrefix "

        if ($ParameterSetName -eq "SkipDocker")
        {
            $installCommand += "-SkipDocker "
        }
        elseif ($ParameterSetName -eq "Staging")
        {
            $installCommand += "-Staging "
        }
        else
        {
            $installCommand += "-DockerPath
            ""$($env:SystemRoot)\System32\docker.exe"" "
        }

        if ($WimName -ne "")
        {
            $installCommand += "-WimPath ""D:\$WimName"" "
        }

        if ($HyperV)
        {
            $installCommand += "-HyperV "
        }
```

```
                #
                # This is required so that Install-ContainerHost.err goes in the right place
                #
                $pwd = "$($env:SystemDrive)\"

                $installCommand += "*>&1 | Tee-Object -FilePath
                ""$($env:SystemDrive)\Install-ContainerHost.log"" -Append"

                Invoke-Expression $installCommand
            }

            Write-Output "Executing Install-ContainerHost.ps1 inside the VM..."
            $wimName = ""
            if ($global:WimSaveMode)
            {
                $wimName = $global:localWimName
            }
            Invoke-Command -VMName $($vm.Name) -Credential $credential -ScriptBlock
            $guestScriptBlock -ArgumentList
            $wimName,$global:ParameterSet,$HyperV,$NATSubnetPrefix

            $scriptFailed = Invoke-Command -VMName $($vm.Name) -Credential $credential
            -ScriptBlock { Test-Path "$($env:SystemDrive)\Install-ContainerHost.err" }

            if ($scriptFailed)
            {
                throw "Install-ContainerHost.ps1 failed in the VM"
            }

            #
            # Cleanup
            #
            if ($global:WimSaveMode)
            {
                Write-Output "Cleaning up temporary WIM VHD"
                $vm | Get-VMHardDiskDrive |? Path -eq $wimVhd.Path | Remove-VMHardDiskDrive
                Remove-Item $wimVhd.Path
            }

            Write-Output "VM $($vm.Name) is ready for use as a container host."
        }
        else
        {
            Write-Output "VM $($vm.Name) will be ready to use as a container host when
            Install-ContainerHost.ps1 completes execution inside the VM."
        }
    }

    Write-Output "See
    https://msdn.microsoft.com/virtualization/windowscontainers/containers_welcome for more
    information about using Containers."
    Write-Output "The source code for these installation scripts is available here:
    https://github.com/Microsoft/Virtualization-Documentation/tree/master/windows-server-conta
    iner-tools"
}
$global:AdminPriviledges = $false
```

```
$global:DockerServiceName = "Docker"

function
Copy-File
{
    [CmdletBinding()]
    param(
        [string]
        $SourcePath,

        [string]
        $DestinationPath
    )

    if ($SourcePath -eq $DestinationPath)
    {
        return
    }

    if (Test-Path $SourcePath)
    {
        Copy-Item -Path $SourcePath -Destination $DestinationPath
    }
    elseif (($SourcePath -as [System.URI]).AbsoluteURI -ne $null)
    {
        if (Test-Nano)
        {
            $handler = New-Object System.Net.Http.HttpClientHandler
            $client = New-Object System.Net.Http.HttpClient($handler)
            $client.Timeout = New-Object System.TimeSpan(0, 30, 0)
            $cancelTokenSource = [System.Threading.CancellationTokenSource]::new()
            $responseMsg = $client.GetAsync([System.Uri]::new($SourcePath),
            $cancelTokenSource.Token)
            $responseMsg.Wait()

            if (!$responseMsg.IsCanceled)
            {
                $response = $responseMsg.Result
                if ($response.IsSuccessStatusCode)
                {
                    $downloadedFileStream = [System.IO.FileStream]::new($DestinationPath,
                    [System.IO.FileMode]::Create, [System.IO.FileAccess]::Write)
                    $copyStreamOp = $response.Content.CopyToAsync($downloadedFileStream)
                    $copyStreamOp.Wait()
                    $downloadedFileStream.Close()
                    if ($copyStreamOp.Exception -ne $null)
                    {
                        throw $copyStreamOp.Exception
                    }
                }
            }
        }
        elseif ($PSVersionTable.PSVersion.Major -ge 5)
        {
            #
            # We disable progress display because it kills performance for large downloads
            (at least on 64-bit PowerShell)
```

```
            #
            $ProgressPreference = 'SilentlyContinue'
            wget -Uri $SourcePath -OutFile $DestinationPath -UseBasicParsing
            $ProgressPreference = 'Continue'
        }
        else
        {
            $webClient = New-Object System.Net.WebClient
            $webClient.DownloadFile($SourcePath, $DestinationPath)
        }
    }
    else
    {
        throw "Cannot copy from $SourcePath"
    }
}


function
Expand-ArchiveNano
{
    [CmdletBinding()]
    param
    (
        [string] $Path,
        [string] $DestinationPath
    )

    [System.IO.Compression.ZipFile]::ExtractToDirectory($Path, $DestinationPath)
}


function
Expand-ArchivePrivate
{
    [CmdletBinding()]
    param
    (
        [Parameter(Mandatory=$true)]
        [string]
        $Path,

        [Parameter(Mandatory=$true)]
        [string]
        $DestinationPath
    )

    $shell = New-Object -com Shell.Application
    $zipFile = $shell.NameSpace($Path)

    $shell.NameSpace($DestinationPath).CopyHere($zipFile.items())

}


function
Get-InstalledContainerImage
```

```
{
    [CmdletBinding()]
    param(
        [Parameter(Mandatory=$true)]
        [string]
        [ValidateNotNullOrEmpty()]
        $BaseImageName
    )

    return Get-ContainerImage |? IsOSImage |? Name -eq $BaseImageName
}


function
Get-Nsmm
{
    [CmdletBinding()]
    param(
        [Parameter(Mandatory=$true)]
        [string]
        [ValidateNotNullOrEmpty()]
        $Destination,

        [string]
        [ValidateNotNullOrEmpty()]
        $WorkingDir = "$env:temp"
    )

    Write-Output "This script uses a third party tool: NSSM. For more information, see
    https://nssm.cc/usage"
    Write-Output "Downloading NSSM..."

    $nssmUri = "https://nssm.cc/release/nssm-2.24.zip"
    $nssmZip = "$($env:temp)\$(Split-Path $nssmUri -Leaf)"

    Write-Verbose "Creating working directory..."
    $tempDirectory = New-Item -ItemType Directory -Force -Path "$($env:temp)\nssm"

    Copy-File -SourcePath $nssmUri -DestinationPath $nssmZip

    Write-Output "Extracting NSSM from archive..."
    if (Test-Nano)
    {
        Expand-ArchiveNano -Path $nssmZip -DestinationPath $tempDirectory.FullName
    }
    elseif ($PSVersionTable.PSVersion.Major -ge 5)
    {
        Expand-Archive -Path $nssmZip -DestinationPath $tempDirectory.FullName
    }
    else
    {
        Expand-ArchivePrivate -Path $nssmZip -DestinationPath $tempDirectory.FullName
    }
    Remove-Item $nssmZip

    Write-Verbose "Copying NSSM to $Destination..."
    Copy-Item -Path "$($tempDirectory.FullName)\nssm-2.24\win64\nssm.exe" -Destination
```

```
    "$Destination"

    Write-Verbose "Removing temporary directory..."
    $tempDirectory | Remove-Item -Recurse
}


function
Test-Admin()
{
    # Get the ID and security principal of the current user account
    $myWindowsID=[System.Security.Principal.WindowsIdentity]::GetCurrent()
    $myWindowsPrincipal=new-object System.Security.Principal.WindowsPrincipal($myWindowsID)

    # Get the security principal for the Administrator role
    $adminRole=[System.Security.Principal.WindowsBuiltInRole]::Administrator

    # Check to see if we are currently running "as Administrator"
    if ($myWindowsPrincipal.IsInRole($adminRole))
    {
        $global:AdminPriviledges = $true
        return
    }
    else
    {
        #
        # We are not running "as Administrator"
        # Exit from the current, unelevated, process
        #
        throw "You must run this script as administrator"
    }
}


function
Test-ContainerProvider()
{
    if (-not (Get-Command Install-ContainerImage -ea SilentlyContinue))
    {
        Wait-Network

        Write-Output "Installing ContainerProvider package..."
        Install-PackageProvider ContainerProvider -Force | Out-Null
    }

    if (-not (Get-Command Install-ContainerImage -ea SilentlyContinue))
    {
        throw "Could not install ContainerProvider"
    }
}


function
Test-Nano()
{
    $EditionId = (Get-ItemProperty -Path 'HKLM:\SOFTWARE\Microsoft\Windows
    NT\CurrentVersion' -Name 'EditionID').EditionId
```

```
    return (($EditionId -eq "NanoServer") -or ($EditionId -eq "ServerTuva"))
}


function
Wait-Network()
{
    $connectedAdapter = Get-NetAdapter |? ConnectorPresent

    if ($connectedAdapter -eq $null)
    {
        throw "No connected network"
    }

    $startTime = Get-Date
    $timeElapsed = $(Get-Date) - $startTime

    while ($($timeElapsed).TotalMinutes -lt 5)
    {
        $readyNetAdapter = $connectedAdapter |? Status -eq 'Up'

        if ($readyNetAdapter -ne $null)
        {
            return;
        }

        Write-Output "Waiting for network connectivity..."
        Start-Sleep -sec 5

        $timeElapsed = $(Get-Date) - $startTime
    }

    throw "Network not connected after 5 minutes"
}


function
Find-DockerImages
{
    [CmdletBinding()]
    param(
        [Parameter(Mandatory=$true)]
        [string]
        [ValidateNotNullOrEmpty()]
        $BaseImageName
    )

    return docker images | Where { $_ -match $BaseImageName.tolower() }
}


function
Start-Docker()
{
    Write-Output "Starting $global:DockerServiceName..."
    if (Test-Nano)
```

```
    {
        Start-ScheduledTask -TaskName $global:DockerServiceName
    }
    else
    {
        Start-Service -Name $global:DockerServiceName
    }
}


function
Stop-Docker()
{
    Write-Output "Stopping $global:DockerServiceName..."
    if (Test-Nano)
    {
        Stop-ScheduledTask -TaskName $global:DockerServiceName

        #
        # ISSUE: can we do this more gently?
        #
        Get-Process $global:DockerServiceName | Stop-Process -Force
    }
    else
    {
        Stop-Service -Name $global:DockerServiceName
    }
}


function
Test-Docker()
{
    $service = $null

    if (Test-Nano)
    {
        $service = Get-ScheduledTask -TaskName $global:DockerServiceName -ErrorAction
        SilentlyContinue
    }
    else
    {
        $service = Get-Service -Name $global:DockerServiceName -ErrorAction SilentlyContinue
    }

    return ($service -ne $null)
}


function
Wait-Docker()
{
    Write-Output "Waiting for Docker daemon..."
    $dockerReady = $false
    $startTime = Get-Date

    while (-not $dockerReady)
```

```
    {
        try
        {
            if (Test-Nano)
            {
                #
                # Nano doesn't support Invoke-RestMethod, we will parse 'docker ps' output
                #
                if ((docker ps 2>&1 | Select-String "error") -ne $null)
                {
                    throw "Docker daemon is not running yet"
                }
            }
            else
            {
                Invoke-RestMethod -Uri http://127.0.0.1:2375/info -Method GET | Out-Null
            }
            $dockerReady = $true
        }
        catch
        {
            $timeElapsed = $(Get-Date) - $startTime

            if ($($timeElapsed).TotalMinutes -ge 1)
            {
                throw "Docker Daemon did not start successfully within 1 minute."
            }

            # Swallow error and try again
            Start-Sleep -sec 1
        }
    }
    Write-Output "Successfully connected to Docker Daemon."
}


function
Write-DockerImageTag()
{
    [CmdletBinding()]
    param(
        [Parameter(Mandatory=$true)]
        [string]
        $BaseImageName
    )

    $dockerOutput = Find-DockerImages $BaseImageName

    if ($dockerOutput.Count -gt 1)
    {
        Write-Output "Base image is already tagged:"
    }
    else
    {
        if ($dockerOutput.Count -lt 1)
        {
            #
```

```
            # Docker restart required if the image was installed after Docker was
            # last started
            #
            Stop-Docker
            Start-Docker

            $dockerOutput = Find-DockerImages $BaseImageName

            if ($dockerOutput.Count -lt 1)
            {
                throw "Could not find Docker image to match '$BaseImageName'"
            }
        }

        if ($dockerOutput.Count -gt 1)
        {
            Write-Output "Base image is already tagged:"
        }
        else
        {
            #
            # Register the base image with Docker
            #
            $imageId = ($dockerOutput -split "\s+")[2]

            Write-Output "Tagging new base image ($imageId)..."

            docker tag $imageId "$($BaseImageName.tolower()):latest"
            Write-Output "Base image is now tagged:"

            $dockerOutput = Find-DockerImages $BaseImageName
        }
    }

    Write-Output $dockerOutput
}

function
Get-Unattend
{
    [CmdletBinding()]
    param(
        [string]
        $Password
    )

    $unattendSource = [xml]@"
<?xml version="1.0" encoding="utf-8"?>
<unattend xmlns="urn:schemas-microsoft-com:unattend">
    <servicing></servicing>
    <settings pass="generalize">
        <component name="Microsoft-Windows-PnpSysprep" processorArchitecture="amd64"
        publicKeyToken="31bf3856ad364e35" language="neutral" versionScope="nonSxS"
        xmlns:wcm="http://schemas.microsoft.com/WMIConfig/2002/State"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
            <PersistAllDeviceInstalls>true</PersistAllDeviceInstalls>
            <DoNotCleanUpNonPresentDevices>true</DoNotCleanUpNonPresentDevices>
```

```
        </component>
        <component name="Microsoft-Windows-Security-SPP" processorArchitecture="amd64"
        publicKeyToken="31bf3856ad364e35" language="neutral" versionScope="nonSxS"
        xmlns:wcm="http://schemas.microsoft.com/WMIConfig/2002/State"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
            <SkipRearm>1</SkipRearm>
        </component>
    </settings>"
    <settings pass="specialize">
        <component name="Microsoft-Windows-Shell-Setup" processorArchitecture="amd64"
        publicKeyToken="31bf3856ad364e35" language="neutral" versionScope="nonSxS"
        xmlns:wcm="http://schemas.microsoft.com/WMIConfig/2002/State"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
            <AutoLogon>
               <Password>
                  <Value>$Password</Value>
                  <PlainText>true</PlainText>
               </Password>
               <Enabled>true</Enabled>
               <LogonCount>999</LogonCount>
               <Username>Administrator</Username>
            </AutoLogon>
            <ComputerName>*</ComputerName>
            <ProductKey>2KNJJ-33Y9H-2GXGX-KMQWH-G6H67</ProductKey>
        </component>
        <component name="Microsoft-Windows-TerminalServices-LocalSessionManager"
        processorArchitecture="amd64" publicKeyToken="31bf3856ad364e35" language="neutral"
        versionScope="nonSxS" xmlns:wcm="http://schemas.microsoft.com/WMIConfig/2002/State"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
             <fDenyTSConnections>false</fDenyTSConnections>
         </component>
         <component name="Microsoft-Windows-TerminalServices-RDP-WinStationExtensions"
         processorArchitecture="amd64" publicKeyToken="31bf3856ad364e35" language="neutral"
         versionScope="nonSxS" xmlns:wcm="http://schemas.microsoft.com/WMIConfig/2002/State"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
             <UserAuthentication>0</UserAuthentication>
         </component>
         <component name="Networking-MPSSVC-Svc" processorArchitecture="amd64"
         publicKeyToken="31bf3856ad364e35" language="neutral" versionScope="nonSxS"
         xmlns:wcm="http://schemas.microsoft.com/WMIConfig/2002/State"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
             <FirewallGroups>
                 <FirewallGroup wcm:action="add" wcm:keyValue="RemoteDesktop">
                     <Active>true</Active>
                     <Profile>all</Profile>
                     <Group>@FirewallAPI.dll,-28752</Group>
                 </FirewallGroup>
             </FirewallGroups>
         </component>
    </settings>
    <settings pass="oobeSystem">
        <component name="Microsoft-Windows-Shell-Setup" processorArchitecture="amd64"
        publicKeyToken="31bf3856ad364e35" language="neutral" versionScope="nonSxS"
        xmlns:wcm="http://schemas.microsoft.com/WMIConfig/2002/State"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
            <OOBE>
                <HideEULAPage>true</HideEULAPage>
```

```
                <HideLocalAccountScreen>true</HideLocalAccountScreen>
                <HideWirelessSetupInOOBE>true</HideWirelessSetupInOOBE>
                <NetworkLocation>Work</NetworkLocation>
                <ProtectYourPC>1</ProtectYourPC>
            </OOBE>
            <UserAccounts>
                <AdministratorPassword>
                    <Value>$Password</Value>
                    <PlainText>True</PlainText>
                </AdministratorPassword>
            </UserAccounts>
        </component>
        <component name="Microsoft-Windows-International-Core" processorArchitecture="amd64"
        publicKeyToken="31bf3856ad364e35" language="neutral" versionScope="nonSxS"
        xmlns:wcm="http://schemas.microsoft.com/WMIConfig/2002/State"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
            <InputLocale>en-us</InputLocale>
            <SystemLocale>en-us</SystemLocale>
            <UILanguage>en-us</UILanguage>
            <UILanguageFallback>en-us</UILanguageFallback>
            <UserLocale>en-us</UserLocale>
        </component>
    </settings>
</unattend>
"@

    return $unattendSource
}

$global:MinimumWimSaveBuild = 10586
$global:MinimumPowerShellBuild = 10240
$global:MinimumSupportedBuild = 9600


function
Approve-Eula
{
    $choiceList = New-Object
    System.Collections.ObjectModel.Collection[System.Management.Automation.Host.ChoiceDescript
    ion]

    $choiceList.Add((New-Object "System.Management.Automation.Host.ChoiceDescription"
    -ArgumentList "&No"))
    $choiceList.Add((New-Object "System.Management.Automation.Host.ChoiceDescription"
    -ArgumentList "&Yes"))

    $eulaText = @"
Before installing and using the Windows Server Technical Preview 4 with Containers virtual
machine you must:
    1.  Review the license terms by navigating to this link: http://aka.ms/tp4/containerseula
    2.  Print and retain a copy of the license terms for your records.
By downloading and using the Windows Server Technical Preview 4 with Containers virtual
machine you agree to such license terms. Please confirm you have accepted and agree to the
license terms.
"@

    return [boolean]$Host.ui.PromptForChoice($null, $eulaText, $choiceList, 0)
```

```
}


function
Test-Version()
{
    $os = Get-WmiObject -Class Win32_OperatingSystem

    if ([int]($os.BuildNumber) -lt $global:MinimumSupportedBuild)
    {
        throw "This script is not supported.  Upgrade to build $global:MinimumSupportedBuild
        or higher."
    }

    if ([int]($os.BuildNumber) -lt $global:MinimumPowerShellBuild)
    {
        Write-Warning "PowerShell Direct is not supported on this version of Windows."
        $global:PowerShellDirectMode = $false
    }

    if ([int]($os.BuildNumber) -lt $global:MinimumWimSaveBuild)
    {
        Write-Warning "Save-ContainerImage is not supported on this version of Windows."
        $global:WimSaveMode = $false
    }

    if (-not (Get-Module Hyper-V))
    {
        throw "Hyper-V must be enabled on this machine."
    }
}


try
{
    Test-Version

    Test-Admin

    if (-not $(Approve-Eula))
    {
        throw "Read and accept the EULA to continue."
    }

    New-ContainerHost
}
catch
{
    Write-Error $_
}
```

TILLEGG

E

RÅDATA FRA GEEKBENCH-TESTER

Host

Kjøring	int single core	int multicore	floating point single core	floating point multicore	Memory score single core	memory score multicore	Geekbench score singlecore	Geekbench score multicore
1	1434	10212	1590	12055	1437	2421	1497	9391
2	1419	10333	1577	12117	1564	2365	1511	9453
3	1439	10291	1591	12175	1437	2394	1499	9465
4	1429	10378	1583	12103	1561	2395	1517	9471
5	1426	10390	1584	12087	1565	2447	1517	9480
6	1431	10310	1592	12192	1454	2440	1500	9488
7	1427	10251	1585	12136	1446	2272	1494	9409
8	1437	10282	1591	12171	1504	2395	1512	9460
9	1424	10271	1592	12139	1572	2415	1520	9447
10	1421	10248	1587	12080	1572	2434	1517	9418
11	1428	10149	1586	12140	1571	2347	1519	9385
12	1426	10170	1589	12153	1508	2299	1507	9389
13	1442	10296	1593	12069	1456	2401	1505	9426
14	1433	10394	1589	12100	1478	2336	1504	9464
15	1431	10313	1590	12180	1491	2399	1506	9477
16	1439	10290	1591	12254	1475	2456	1507	9508
17	1419	10277	1591	12128	1451	2419	1494	9445
18	1426	10365	1585	12134	1522	2435	1508	9486
19	1422	10213	1580	12064	1560	2328	1512	9376
20	1431	10328	1583	12201	1566	2327	1518	9477
21	1424	10301	1580	12138	1565	2431	1514	9461
22	1427	10331	1587	12015	1560	2301	1517	9398
23	1428	10335	1582	12122	1566	2356	1517	9454
24	1427	10257	1586	12148	1563	2418	1517	9445
25	1428	10363	1587	12147	1534	2431	1512	9490
26	1438	10360	1586	12231	1442	2344	1498	9505
27	1424	10267	1590	12208	1534	2429	1512	9475
28	1440	10287	1581	12119	1561	2392	1520	9440
29	1437	10274	1591	12169	1511	2405	1513	9458
30	1438	10301	1591	12165	1516	2432	1514	9472
Gjennomsnitt	1429,833333	10294,56667	1587	12138	1518,066667	2388,8	1509,933333	9450,433333
Standardavvik	6,544717972	59,89973423	4,266954013	53,26576179	49,3404313	49,47266748	7,969482021	36,39314385

VM

Kjøring	int single core	int multicore	floating point single core	floating point multicore	Memory score single core	memory score multicore	Geekbench score singlecore	Geekbench score multicore
1	1419	8828	1571	10736	1379	2350	1471	8295
2	1411	9039	1576	11300	1520	2311	1498	8597
3	1402	8965	1573	11304	1541	2238	1498	8555
4	1404	9309	1575	10791	1550	2388	1501	8517
5	1400	9346	1575	10923	1553	2413	1500	8590
6	1406	9493	1570	11368	1547	2255	1499	8795
7	1417	9448	1577	11362	1532	2382	1504	8800
8	1406	9134	1569	11154	1445	2267	1479	8568
9	1421	9743	1579	11022	1530	2356	1506	8777
10	1404	9430	1570	10764	1527	2355	1495	8548
11	1415	9551	1578	11365	1534	2278	1504	8822
12	1407	9428	1569	11558	1554	2332	1501	8860
13	1417	9398	1589	11159	1539	2293	1510	8681
14	1407	9470	1574	11265	1546	2411	1501	8776
15	1421	9444	1582	11365	1537	2200	1508	8763
16	1405	9348	1567	10447	1546	2364	1498	8390
17	1407	9166	1569	11018	1548	2215	1500	8516
18	1418	9307	1576	11007	1531	2239	1503	8573
19	1405	9272	1569	11129	1535	2226	1496	8605
20	1414	9422	1582	10944	1532	2359	1504	8618
21	1405	9260	1572	10843	1536	2364	1498	8514
22	1419	9032	1583	11664	1532	2284	1507	8735
23	1405	9192	1566	11227	1543	2329	1497	8633
24	1413	9681	1583	11356	1551	2364	1508	8887
25	1403	9399	1561	11252	1552	2392	1496	8738
26	1415	9215	1576	11518	1388	2257	1474	8744
27	1404	9458	1571	11371	1549	2334	1499	8798
28	1408	9512	1571	11232	1537	2395	1499	8776
29	1401	9577	1567	11144	1551	2307	1497	8749
30	1415	9574	1579	11517	1540	2390	1505	8914
Gjennomsnitt	1409,8	9348,033333	1573,966667	11170,16667	1526,833333	2321,6	1498,533333	8671,133333
Standardavvik	6,514704852	210,0540954	6,155924163	277,5379543	43,55502531	62,91461099	9,069590468	148,1368736

Container

Kjøring	int single core	int multicore	floating point single core	floating point multicore	Memory score single core	memory score multicore	Geekbench score singlecore	Geekbench score multicore
1	1412	8740	1578	10410	1540	2409	1504	8141
2	1414	8913	1580	10507	1414	2318	1480	8231
3	1399	8929	1570	10826	1542	2268	1496	8355
4	1401	8937	1571	10143	1549	2380	1498	8108
5	1410	9063	1577	10770	1445	2381	1483	8409
6	1400	8790	1575	10587	1401	2025	1470	8155
7	1409	9040	1576	10918	1543	2415	1502	8466
8	1400	9302	1570	10942	1533	2277	1494	8553
9	1413	8801	1577	10397	1550	2411	1506	8161
10	1401	9164	1563	11051	1551	2364	1495	8558
11	1412	8802	1580	10571	1535	2414	1503	8232
12	1403	9116	1572	11256	1553	2347	1500	8618
13	1410	9264	1579	10708	1539	2419	1503	8472
14	1402	9559	1573	10893	1536	2369	1497	8654
15	1407	9397	1577	10400	1556	2411	1504	8401
16	1406	8571	1565	10575	1525	2152	1493	8088
17	1416	8804	1575	10563	1410	2293	1478	8205
18	1402	8854	1571	11375	1541	1721	1497	8435
19	1394	9390	1579	10810	1559	2142	1501	8508
20	1405	9363	1563	10375	1397	1687	1466	8232
21	1392	9039	1581	11213	1538	2245	1496	8549
22	1402	9162	1564	11083	1530	1938	1492	8485
23	1396	9294	1578	10693	1534	2319	1496	8458
24	1402	9125	1575	10811	1532	2380	1497	8450
25	1414	8985	1577	10670	1164	2198	1429	8301
26	1403	9433	1570	11451	1537	1836	1496	8720
27	1411	9345	1581	10675	1159	2412	1428	8490
28	1402	9347	1569	10994	1382	2370	1464	8610
29	1416	8934	1579	10687	1399	2415	1477	8331
30	1406	9001	1572	10526	1394	2133	1470	8237
Gjennomsnitt	1405,333333	9082,133333	1573,9	10762,66667	1479,6	2248,3	1487,166667	8387,1
Standardavvik	6,466536318	247,0928533	5,390604854	312,6029026	106,0993679	210,1938596	19,99841948	176,2489129

Host med en VM (idle)

Kjøring	int single core	int multicore	floating point single core	floating point multicore	Memory score single core	memory score multicore	Geekbench score singlecore	Geekbench score multicore
1	1423	10225	1580	12087	1559	2420	1513	9408
2	1427	9980	1582	12139	1504	2397	1504	9327
3	1427	10301	1578	12126	1556	2420	1513	9454
4	1429	10207	1586	11836	1511	2413	1508	9299
5	1421	10192	1580	12010	1559	2429	1512	9366
6	1433	10177	1587	12095	1544	2400	1516	9388
7	1423	10147	1576	12044	1556	2395	1510	9355
8	1430	10226	1591	12067	1517	2411	1511	9399
9	1424	10174	1584	12075	1551	2409	1513	9381
10	1439	10169	1587	12126	1547	2413	1519	9400
11	1425	10110	1581	11910	1555	2396	1513	9287
12	1437	10336	1588	12058	1559	2418	1521	9441
13	1422	10280	1581	12013	1557	2372	1512	9391
14	1426	10200	1582	11977	1495	2185	1502	9307
15	1423	9985	1583	11987	1557	2391	1513	9267
16	1421	10290	1581	12105	1561	2299	1513	9417
17	1436	10307	1592	12096	1514	2418	1514	9444
18	1423	10215	1583	12033	1559	2411	1514	9381
19	1435	10266	1589	12219	1548	2392	1519	9472
20	1423	10275	1578	12198	1514	2427	1503	9474
21	1437	10261	1592	12213	1544	2387	1520	9467
22	1425	10407	1586	11997	1557	2424	1515	9446
23	1437	10217	1585	12165	1562	2376	1521	9428
24	1423	10174	1582	12085	1559	2349	1513	9373
25	1439	10186	1592	12066	1525	2395	1517	9379
26	1425	10209	1582	12051	1553	2305	1513	9365
27	1431	10232	1591	12101	1486	2406	1506	9414
28	1426	10192	1581	12127	1556	2428	1514	9413
29	1432	10287	1590	11980	1564	2381	1521	9383
30	1419	10281	1579	12034	1564	2435	1512	9413
Gjennomsnitt	1428,033333	10216,93333	1584,3	12067,33333	1543,1	2390,066667	1513,166667	9391,3
Standardavvik	6,076996385	88,74486224	4,691518393	84,32217908	22,79798842	50,34428594	5,038358609	54,026909

Host med en vm og 5 containere (idle)

Kjøring	int single core	int multicore	floating point single core	floating point multicore	Memory score single core	memory score multicore	Geekbench score singlecore	Geekbench score multicore
1	1429	9876	1586	11778	1544	2389	1514	9139
2	1421	9955	1573	11830	1554	2388	1508	9191
3	1439	9947	1583	11764	1539	2367	1516	9157
4	1423	9909	1582	11891	1562	2389	1514	9197
5	1431	9937	1579	11929	1532	2419	1510	9230
6	1426	9923	1577	11821	1560	2362	1513	9170
7	1435	9971	1590	11774	1533	2385	1516	9175
8	1423	9916	1580	11902	1559	2426	1513	9212
9	1422	9897	1575	11721	1555	2404	1509	9128
10	1429	9970	1583	11733	1549	2362	1514	9153
11	1426	9845	1579	11795	1540	2290	1510	9114
12	1436	10092	1589	11897	1549	2365	1519	9268
13	1420	10078	1575	11657	1559	2320	1509	9158
14	1433	9941	1587	11711	1547	2380	1517	9136
15	1422	9885	1580	11694	1553	2349	1511	9101
16	1420	9950	1583	11756	1557	2328	1512	9148
17	1434	9611	1576	11793	1550	2403	1514	9042
18	1438	9953	1587	11781	1519	2378	1513	9169
19	1426	10110	1582	11656	1561	2392	1515	9184
20	1436	9963	1584	11867	1555	2367	1519	9205
21	1425	10015	1584	11698	1560	2396	1515	9164
22	1433	9956	1590	11823	1528	2331	1514	9177
23	1422	10016	1582	11733	1556	2413	1512	9182
24	1434	9963	1586	11678	1555	2433	1519	9143
25	1423	9920	1573	11711	1560	2394	1510	9131
26	1430	10022	1592	11798	1559	2396	1520	9207
27	1424	10079	1583	11799	1560	2335	1514	9218
28	1436	10032	1586	11784	1565	2429	1521	9212
29	1427	10060	1578	11662	1565	2340	1515	9156
30	1435	9937	1591	11817	1561	2400	1522	9181
Gjennomsnitt	1428,6	9957,633333	1582,5	11775,1	1551,533333	2377,666667	1514,266667	9168,266667
Standardavvik	5,97465913	93,62967527	5,328841499	75,58614633	11,62557624	34,73379884	3,694668417	43,53986151

Host med 5 vms (idle)

Kjøring	int single core	int multicore	floating point single core	floating point multicore	Memory score single core	memory score multicore	Geekbench score singlecore	Geekbench score multicore
1	1430	9931	1581	11945	1534	2375	1511	9225
2	1420	9946	1577	11919	1540	2306	1506	9207
3	1417	9841	1580	11832	1543	2391	1507	9147
4	1416	9969	1579	11839	1535	2377	1505	9198
5	1423	10015	1576	11787	1500	2295	1499	9179
6	1418	10138	1571	11794	1543	2342	1504	9241
7	1421	10122	1579	11852	1534	2334	1506	9256
8	1430	10121	1585	11763	1509	2323	1507	9218
9	1414	10073	1576	11762	1513	2343	1498	9202
10	1427	10037	1581	11717	1491	2376	1501	9176
11	1416	9971	1575	11765	1539	2332	1504	9160
12	1432	9986	1583	11811	1524	2340	1510	9186
13	1415	9931	1574	11692	1537	2398	1503	9128
14	1431	9963	1584	11785	1514	2308	1508	9160
15	1424	9875	1587	11750	1499	2273	1504	9104
16	1418	9872	1577	11747	1539	2295	1505	9106
17	1433	9846	1573	11810	1496	2342	1501	9130
18	1416	9897	1580	11759	1518	2370	1502	9136
19	1426	9936	1579	11714	1524	2351	1506	9130
20	1417	9980	1579	11732	1539	2289	1506	9142
21	1426	9873	1576	11759	1537	2316	1508	9116
22	1414	9975	1577	11746	1556	2286	1507	9145
23	1430	10029	1584	11860	1502	2324	1506	9220
24	1418	10025	1574	11759	1542	2387	1505	9191
25	1432	10000	1587	11847	1503	2309	1508	9200
26	1415	9992	1571	11885	1541	2224	1502	9195
27	1411	9993	1576	11726	1534	2276	1501	9142
28	1426	9945	1583	11359	1541	2347	1511	8991
29	1416	9997	1583	11776	1525	2370	1504	9183
30	1431	9894	1585	11868	1530	2363	1512	9177
Gjennomsnitt	1422,1	9972,433333	1579,066667	11778,66667	1526,066667	2332,066667	1505,233333	9166,366667
Standardavvik	6,814538211	78,17164286	4,494696747	100,6020955	17,34822526	40,96334218	3,46094798	51,88015589

Host med 5 vms under load (1 cpu pr)

Kjøring	int single core	int multicore	floating point single core	floating point multicore	Memory score single core	memory score multicore	Geekbench score singlecore	Geekbench score multicore
1	1429	9876	1586	11778	1544	2389	1514	4363
2	1421	9955	1573	11830	1554	2388	1508	4661
3	1439	9947	1583	11764	1539	2367	1516	5190
4	1423	9909	1582	11891	1562	2389	1514	4467
5	1431	9937	1579	11929	1532	2419	1510	4281
6	1426	9923	1577	11821	1560	2362	1513	4238
7	1435	9971	1590	11774	1533	2385	1516	4326
8	1423	9916	1580	11902	1559	2426	1513	4045
9	1422	9897	1575	11721	1555	2404	1509	4078
10	1429	9970	1583	11733	1549	2362	1514	4309
11	1426	9845	1579	11795	1540	2290	1510	3977
12	1436	10092	1589	11897	1549	2365	1519	4259
13	1420	10078	1575	11657	1559	2320	1509	4116
14	1433	9941	1587	11711	1547	2380	1517	4292
15	1422	9885	1580	11694	1553	2349	1511	4050
16	1420	9950	1583	11756	1557	2328	1512	4196
17	1434	9611	1576	11793	1550	2403	1514	3960
18	1438	9953	1587	11781	1519	2378	1513	4229
19	1426	10110	1582	11656	1561	2392	1515	4030
20	1436	9963	1584	11867	1555	2367	1519	4322
21	1425	10015	1584	11698	1560	2396	1515	4212
22	1433	9956	1590	11823	1528	2331	1514	4282
23	1422	10016	1582	11733	1556	2413	1512	4233
24	1434	9963	1586	11678	1555	2433	1519	4310
25	1423	9920	1573	11711	1560	2394	1510	4043
26	1430	10022	1592	11798	1559	2396	1520	4237
27	1424	10079	1583	11799	1560	2335	1514	4088
28	1436	10032	1586	11784	1565	2429	1521	4313
29	1427	10060	1578	11662	1565	2340	1515	4299
30	1435	9937	1591	11817	1561	2400	1522	4225
Gjennomsnitt	1428,6	9957,633333	1582,5	11775,1	1551,533333	2377,666667	1514,266667	4254,366667
Standardavvik	5,97465913	93,62967527	5,328841499	75,58614633	11,62557624	34,73379884	3,694668417	231,3202711

Host under load

Kjøring	int single core	int multicore	floating point single core	floating point multicore	Memory score single core	memory score multicore	Geekbench score singlecore	Geekbench score multicore
1	1367	10058	1550	11442	977	1880	1362	8976
2	1351	9823	1540	11405	958	1787	1348	8848
3	1360	9778	1536	11597	1060	1986	1370	8947
4	1347	9568	1540	11551	957	1788	1346	8805
5	1363	9739	1542	11651	1054	2001	1372	8956
6	1351	9793	1543	11282	962	1788	1350	8787
7	1362	9493	1546	11604	1043	2082	1371	8855
8	1365	9841	1541	11301	1062	1995	1374	8855
9	1363	9766	1540	11233	1053	1991	1371	8797
10	1362	9646	1542	11217	1073	1990	1376	8743
11	1368	9670	1537	11520	1059	1991	1373	8874
12	1363	9761	1542	11783	1057	1989	1373	9015
13	1363	9563	1541	11254	1056	1996	1372	8726
14	1353	9790	1541	11387	971	1787	1351	8828
15	1363	9639	1545	11398	1055	1998	1374	8814
16	1354	9857	1536	11489	940	1830	1344	8904
17	1354	9861	1545	11528	962	1795	1352	8914
18	1358	9795	1543	11503	1047	1996	1369	8918
19	1355	9891	1541	11576	920	1785	1342	8943
20	1365	9688	1541	11397	1044	1992	1371	8832
21	1356	9897	1540	11124	1001	1915	1358	8791
22	1372	9845	1555	11525	1050	1959	1380	8939
23	1362	9657	1540	11524	1059	2005	1372	8873
24	1366	9697	1542	11297	1045	1978	1372	8793
25	1349	9460	1542	11538	976	1795	1351	8758
26	1362	9720	1537	11657	1080	1996	1375	8950
27	1355	9869	1542	11567	959	1791	1350	8932
28	1383	9825	1554	12014	1112	2028	1397	9141
29	1386	10009	1547	11809	1074	2033	1388	9133
30	1375	9792	1559	11692	1046	1954	1382	8984
Gjennomsnitt	1361,766667	9759,7	1543	11495,5	1023,733333	1930,033333	1366,2	8887,7
Standardavvik	9,065597469	136,369718	5,394761188	193,720159	50,90757909	96,82386544	13,85740186	102,191217

Host med en vm med 5 container under load

Kjøring	int single core	int multicore	floating point single core	floating point multicore	Memory score single core	memory score multicore	Geekbench score singlecore	Geekbench score multicore
1	1145	5784	1156	6528	693	1630	1059	5250
2	859	4850	997	5739	598	1313	862	4498
3	884	4825	1025	6199	706	1612	904	4732
4	898	4799	1024	5774	691	1637	907	4556
5	886	4890	1002	5707	703	1601	895	4559
6	852	5322	996	6212	566	1318	852	4877
7	909	4924	1010	5619	696	1578	906	4532
8	864	4774	1008	5640	612	1333	871	4432
9	881	5139	1049	6382	689	1556	909	4919
10	907	4778	1028	5920	694	1585	912	4596
11	910	4932	1096	6355	645	1573	931	4829
12	1015	5314	1109	5968	621	1375	973	4787
13	965	5269	1105	6295	659	1602	959	4946
14	1101	5861	1250	7056	811	1752	1102	5517
15	1186	6523	1211	7244	749	1474	1108	5801
16	1241	7121	1336	7429	845	1710	1199	6162
17	1028	5962	1218	7374	763	1421	1051	5618
18	1099	6101	1345	7496	822	1736	1142	5786
19	1141	6251	1291	7089	834	1777	1139	5691
20	1152	5938	1259	6739	635	1377	1091	5346
21	899	5551	1019	6573	672	1571	901	5163
22	922	5078	1017	5994	666	1534	908	4735
23	887	4933	1012	6042	648	1526	889	4695
24	937	5186	1009	6465	658	1334	910	4927
25	876	5152	1036	5963	628	1338	890	4713
26	882	5005	1002	5883	657	1546	885	4664
27	920	5126	1034	5890	651	1332	911	4672
28	904	5032	1140	5958	750	1642	967	4724
29	1016	5215	1198	6085	631	1569	1011	4833
30	960	5483	1250	7187	678	1513	1019	5370
Gjennomsnitt	970,8666667	5370,6	1107,733333	6360,166667	689,0333333	1528,833333	968,7666667	4997,666667
Standardavvik	112,8904667	581,7621033	113,8216794	583,4513887	70,5053108	138,3947586	97,95589554	461,5145482

Host med 5 vms under load (8 cores pr)

Kjøring	int single core	int multicore	floating point single core	floating point multicore	Memory score single core	memory score multicore	Geekbench score singlecore	Geekbench score multicore
1	1088	6256	1295	8318	719	1525	1097	6134
2	1146	6481	1370	8374	683	1561	1143	6254
3	1206	6831	1377	8281	761	1556	1185	6356
4	1193	6939	1357	8442	763	1569	1172	6466
5	1190	7238	1362	8649	756	1581	1172	6671
6	1167	7188	1364	8552	673	1572	1147	6610
7	1193	7199	1360	8341	747	1579	1170	6531
8	1210	7199	1367	8631	771	1570	1185	6646
9	1183	7193	1372	8735	676	1572	1157	6685
10	1166	7152	1385	8678	713	1568	1163	6645
11	1161	7256	1358	8663	697	1564	1147	6680
12	1180	7203	1359	8594	703	1552	1156	6629
13	1186	7177	1370	8603	754	1562	1173	6624
14	1166	7153	1357	8464	698	1583	1148	6563
15	1174	7253	1380	8705	679	1576	1157	6698
16	1187	7173	1380	8621	756	1574	1178	6632
17	1178	7175	1364	8583	687	1575	1154	6618
18	1174	7182	1386	8587	697	1572	1163	6622
19	1207	7177	1364	8435	773	1559	1183	6556
20	1179	7246	1372	8647	683	1585	1157	6674
21	1206	7214	1380	8562	765	1567	1187	6623
22	1173	7001	1369	8577	712	1559	1159	6543
23	1182	7083	1373	8534	713	1560	1164	6558
24	1202	7127	1377	8439	761	1561	1183	6538
25	1206	6516	1394	9281	776	1573	1195	6633
26	1172	7236	1376	8621	678	1590	1154	6660
27	1187	7005	1371	8659	761	1577	1175	6581
28	1189	7211	1376	8637	758	1581	1177	6655
29	1175	7200	1386	8570	707	1578	1165	6623
30	1172	7178	1367	8680	711	1553	1157	6653
Gjennomsnitt	1179,933333	7081,4	1368,933333	8582,1	724,3666667	1568,466667	1164,1	6578,7
Standardavvik	23,21404648	247,3274529	16,90324052	177,8452155	35,46390427	12,74344546	18,64061195	127,4219572

TILLEGG

F

RÅDATA FRA NETTVERKSTESTING

kjøring	Host	VM	Container
1	785.174	759.858	771.137
2	684.884	797.236	777.005
3	722.119	794.662	738.265
4	718.297	797.605	754.249
5	684.954	797.518	751.629
6	670.599	791.758	768.849
7	708.93	794.393	749.702
8	696.534	793.262	726.845
9	799.714	809.618	176.462
10	803.419	766.442	771.458
11	791.465	775.037	733.41
12	780.979	772.028	765.275
13	797.354	757.363	757.173
14	804.414	772.028	756.269
15	786.851	735.505	766.216
16	782.186	742.933	739.555
17	791.465	733.109	759.386
18	790.137	730.385	756.963
19	771.351	757.173	780.951
20	776.925	731.545	688.296
21	765.67	791.024	703.264
22	785.401	774.075	713.856
23	702.289	664.203	767.787
24	787.76	781.714	634.9
25	774.235	683.137	791.635
26	798.053	759.309	767.493
27	813.012	753.502	788.439
28	789.036	791.548	772.08
29	798.735	766.931	761.229
30	708.628	790.954	774.436
Gjennomsnitt	762.3523333	765.5285	732.1404667
Avvik	43.03	33.11	107.97
Max	813.012	809.618	791.635
Min	670.599	664.203	176.462

TILLEGG

G

RÅDATA FRA I/O-TESTING

Host random IO

kjøring	mb/s write	write iops	write latency	mb/s read	read iops	read latency
1	10.21	163.33	97.965	11.54	184.69	86.616
2	10.41	166.53	96.072	11.52	184.35	86.834
3	10.53	168.42	94.993	11.52	184.36	86.819
4	10.45	167.19	95.691	11.49	183.88	86.994
5	10.50	167.98	95.271	11.59	185.42	86.361
6	10.47	167.56	95.486	11.46	183.29	87.311
7	10.47	167.59	95.463	11.65	186.42	85.818
8	10.39	166.26	96.222	11.43	182.96	87.468
9	10.46	167.33	95.614	11.58	185.24	86.399
10	10.43	166.88	95.875	11.51	184.22	86.851
11	10.52	168.25	95.093	11.65	186.47	85.813
12	10.24	163.86	97.64	11.47	183.58	87.2
13	10.46	167.42	95.566	11.73	187.67	85.248
14	10.58	169.23	94.532	11.68	186.86	85.644
15	10.48	167.63	95.439	11.6	185.56	86.27
16	10.57	169.15	94.588	11.62	185.96	86.019
17	10.45	167.27	95.665	11.52	184.27	86.84
18	10.63	170.07	94.068	11.56	184.94	86.539
19	10.50	167.92	95.288	11.43	182.86	87.481
20	10.71	171.4	93.344	11.52	184.24	86.88
21	10.49	167.79	95.357	11.48	183.62	87.168
22	10.62	169.88	94.182	11.59	185.44	86.284
23	10.36	165.83	96.482	11.51	184.15	86.932
24	10.49	167.77	95.366	11.57	185.18	86.386
25	10.27	164.36	97.339	11.33	181.31	88.251
26	10.42	166.7	95.978	11.45	183.26	87.375
27	10.33	165.29	96.785	11.41	182.59	87.628
28	10.38	166	96.378	11.43	182.94	87.515
29	10.41	166.6	96.031	11.39	182.27	87.794
30	10.45	167.26	95.822	11.59	185.48	86.275
Gjennomsnitt	10.456	167.2916667	95.65316667	11.52733333	184.4493333	86.7671
avvik	0.11	1.72	0.99	0.09	1.44	0.68

VM random IO

kjøring	mb/s write	write iops	write latency	mb/s read	read iops	read latency
1	10.09	161.4	99.125	11.34	181.36	88.207
2	10.49	167.85	95.314	11.54	184.57	86.72
3	10.46	167.34	95.608	11.54	184.62	86.683
4	10.48	167.66	95.421	11.56	184.9	86.548
5	10.52	168.32	95.049	11.52	184.25	86.843
6	10.44	166.99	96.07	11.66	186.63	85.768
7	10.42	166.76	95.938	11.54	184.68	86.668
8	10.46	167.29	95.64	11.54	184.61	86.645
9	10.54	168.63	94.884	11.46	183.43	87.22
10	10.43	166.87	95.87	11.42	182.74	87.543
11	10.48	167.69	95.536	11.4	182.45	87.725
12	10.19	163.09	98.091	11.37	181.97	87.935
13	10.54	168.61	94.898	11.47	183.55	87.188
14	10.45	167.13	95.726	11.43	182.82	87.535
15	10.47	167.6	95.458	11.53	184.43	86.784
16	10.51	168.12	95.171	11.62	185.96	86.021
17	10.42	166.69	95.985	11.53	184.54	86.705
18	10.51	168.2	95.132	11.51	184.13	86.941
19	10.46	167.37	95.586	11.5	184.01	86.98
20	10.42	166.79	95.911	11.56	184.95	86.464
21	10.43	166.85	95.878	11.49	183.88	87.03
22	10.51	168.17	95.145	11.4	182.39	87.758
23	10.33	165.25	96.816	11.28	180.51	88.687
24	10.40	166.46	96.11	11.47	183.54	87.181
25	10.37	165.97	96.39	11.42	182.69	87.578
26	10.43	166.87	96.003	11.49	183.85	87.048
27	10.41	166.64	96.011	11.46	183.33	87.288
28	10.57	169.16	94.583	11.42	182.78	87.556
29	10.46	167.36	95.604	11.41	182.62	87.589
30	10.45	168.29	95.063	11.41	182.5	87.69
Gjennomsnitt	10.438	167.0473333	95.80053333	11.47633333	183.623	87.15093333
avvik	0.10	1.54	0.90	0.08	1.29	0.62

Container random IO

kjøring	mb/s write	write iops	write latency	mb/s read	read iops	read latency
1	10.10	161.57	99.027	11.32	181.16	88.318
2	10.42	166.68	95.991	11.33	181.25	88.261
3	10.32	165.05	96.946	11.32	181.11	88.37
4	10.35	165.65	96.792	11.32	181.19	88.314
5	10.37	165.96	96.406	11.36	181.69	88.041
6	10.32	165.09	96.917	11.32	181.1	88.384
7	10.37	165.89	96.44	11.42	182.78	87.562
8	10.30	164.74	97.103	11.46	183.29	87.28
9	10.31	164.91	97.017	11.31	180.96	88.415
10	10.32	165.17	96.861	11.34	181.51	88.138
11	10.37	165.96	96.413	11.28	180.43	88.665
12	10.29	164.65	97.165	11.44	182.98	87.458
13	10.33	165.34	96.768	11.31	180.93	88.453
14	10.36	165.69	96.558	11.26	180.13	88.886
15	10.30	164.77	97.097	11.39	182.19	87.841
16	10.29	164.66	97.17	11.34	181.39	88.232
17	10.36	165.75	96.526	11.34	181.39	88.215
18	10.32	165.06	96.927	11.27	180.26	88.751
19	10.28	164.44	97.289	11.43	182.9	87.483
20	10.36	165.78	96.509	11.34	181.39	88.188
21	10.43	166.83	95.893	11.44	183.01	87.458
22	10.34	165.41	96.721	11.38	182.06	87.896
23	10.40	166.46	96.123	11.37	181.87	88.014
24	10.38	166.08	96.337	11.44	183.07	87.41
25	10.29	164.61	97.14	11.32	181.18	88.352
26	10.37	165.9	96.435	11.34	181.4	88.208
27	10.40	166.45	96.359	11.46	183.41	87.247
28	10.49	167.81	95.332	11.37	181.88	87.994
29	10.34	165.43	96.708	11.29	180.63	88.585
30	10.40	166.37	96.157	11.34	181.4	88.251
Gjennomsnitt	10.34266667	165.472	96.70423333	11.355	181.6646667	88.089
avvik	0.07	1.05	0.61	0.06	0.90	0.44

Host Sequential IO

kjøring	mb/s write	write iops	write latency	mb/s read	read iops	read latency
1	78.34	1253.37	12.764	78.5	1256.02	12.737
2	78.31	1252.94	12.768	78.53	1256.54	12.732
3	78.38	1254.08	12.757	78.52	1256.37	12.733
4	78.33	1253.28	12.765	78.58	1257.27	12.724
5	78.57	1257.14	12.725	78.65	1258.37	12.713
6	78.39	1254.22	12.755	78.58	1257.27	12.724
7	78.31	1253.01	12.768	78.55	1256.84	12.729
8	78.32	1253.06	12.767	78.56	1256.92	12.728
9	78.35	1253.6	12.762	78.66	1258.48	12.712
10	78.47	1255.54	12.742	78.54	1256.57	12.732
11	78.37	1253.96	12.758	78.55	1256.74	12.73
12	78.40	1254.47	12.753	78.62	1257.98	12.717
13	78.33	1253.3	12.765	78.62	1257.97	12.717
14	77.99	1247.91	12.82	78.56	1256.96	12.727
15	78.39	1254.23	12.755	78.59	1257.38	12.723
16	78.39	1254.2	12.755	78.58	1257.32	12.724
17	78.29	1252.57	12.772	78.55	1256.79	12.729
18	78.31	1253.01	12.768	78.54	1256.61	12.731
19	78.37	1253.88	12.759	78.62	1257.87	12.718
20	78.27	1252.31	12.775	78.6	1257.55	12.721
21	78.44	1255.06	12.747	78.62	1257.96	12.717
22	78.33	1253.24	12.765	78.55	1256.85	12.729
23	78.39	1254.31	12.754	78.52	1256.36	12.733
24	78.29	1252.57	12.772	78.54	1256.68	12.73
25	78.48	1255.66	12.741	78.54	1256.56	12.732
26	78.50	1255.96	12.738	78.52	1256.36	12.733
27	78.25	1252.01	12.778	78.51	1256.09	12.736
28	78.42	1254.77	12.75	78.63	1258.01	12.717
29	78.32	1253.05	12.767	78.52	1256.33	12.734
30	78.16	1250.51	12.793	78.51	1256.14	12.736
Gjennomsnitt	78.34866667	1253.574	12.76193333	78.56533333	1257.038667	12.7266
avvik	0.10	1.66	0.02	0.04	0.70	0.01

VM Sequential IO

kjøring	mb/s write	write iops	write latency	mb/s read	read iops	read latency
1	78.17	1250.77	12.792	77.02	1232.26	13.004
2	78.30	1252.78	12.77	76.73	1227.66	13.031
3	78.04	1248.69	12.828	74.44	1191.07	13.431
4	78.45	1255.14	12.746	76.53	1224.51	13.05
5	78.24	1251.88	12.779	75.36	1205.75	13.268
6	78.02	1248.27	12.816	76.22	1219.56	13.118
7	78.05	1248.87	12.81	75.54	1208.72	13.235
8	78.13	1250.01	12.802	75.81	1213.04	13.188
9	78.10	1249.58	12.803	76.22	1219.6	13.117
10	78.29	1252.66	12.771	76.04	1216.69	13.149
11	78.35	1253.64	12.761	75.58	1209.2	13.229
12	78.10	1249.57	12.803	76.2	1219.23	13.127
13	78.26	1252.19	12.776	76.85	1229.58	13.014
14	78.21	1251.39	12.784	76.7	1227.27	13.035
15	78.28	1252.55	12.773	75.88	1214.1	13.176
16	78.48	1255.65	12.741	77	1232.03	12.994
17	78.14	1250.27	12.795	75.66	1210.55	13.215
18	78.04	1248.64	12.812	75.98	1215.72	13.163
19	78.51	1256.09	12.736	76.07	1217.19	13.143
20	78.18	1250.9	12.789	76.87	1229.95	13.007
21	78.39	1254.19	12.756	75.9	1214.34	13.174
22	78.39	1254.23	12.755	74.29	1188.7	13.458
23	78.24	1251.87	12.779	75.03	1200.53	13.326
24	77.96	1247.32	12.826	76.3	1220.86	13.104
25	78.38	1254.06	12.757	77.02	1232.33	12.982
26	78.46	1255.29	12.744	74.33	1189.22	13.452
27	78.07	1249.08	12.808	76.1	1217.65	13.139
28	78.53	1256.49	12.732	76.94	1231.01	12.996
29	78.27	1252.29	12.775	75.65	1210.36	13.187
30	78.37	1253.9	12.759	74.85	1197.57	13.355
Gjennomsnitt	78.24666667	1251.942	12.77926667	75.97033333	1215.541667	13.16223333
avvik	0.16	2.52	0.03	0.78	12.56	0.14

Container Sequential IO

kjøring	mb/s write	write iops	write latency	mb/s read	read iops	read latency
1	78.30	1252.83	12.77	77.92	1246.72	12.832
2	78.42	1254.66	12.751	76.12	1217.89	13.135
3	78.23	1251.62	12.782	74.89	1198.27	13.343
4	78.46	1255.37	12.744	74.67	1194.7	13.39
5	78.27	1252.35	12.774	74.32	1189.06	13.454
6	78.25	1252.06	12.777	76.2	1219.27	13.12
7	78.53	1256.48	12.732	76.28	1220.41	13.108
8	78.39	1254.3	12.754	76.57	1225.14	13.058
9	78.21	1251.38	12.784	76.31	1220.89	13.103
10	78.28	1252.52	12.772	75.19	1202.99	13.298
11	78.07	1249.11	12.807	75.64	1210.29	13.238
12	77.93	1246.82	12.831	77.68	1242.91	12.873
13	78.10	1249.67	12.802	74.27	1188.26	13.464
14	78.45	1255.18	12.745	75.51	1208.1	13.24
15	78.34	1253.37	12.764	75.51	1208.2	13.241
16	78.49	1255.77	12.739	76.02	1216.29	13.153
17	78.35	1253.6	12.762	76.14	1218.17	13.141
18	78.27	1252.34	12.774	75.59	1209.41	13.227
19	77.99	1247.9	12.82	74.61	1193.73	13.438
20	78.42	1254.72	12.75	75.64	1210.25	13.218
21	78.08	1249.23	12.806	74.85	1197.57	13.359
22	78.10	1249.68	12.802	73.97	1183.54	13.517
23	78.49	1255.88	12.738	78.04	1248.64	12.811
24	78.42	1254.77	12.75	76.11	1217.73	13.121
25	78.39	1254.3	12.754	77.67	1242.65	12.874
26	78.23	1251.63	12.782	77.93	1246.93	12.859
27	78.15	1250.41	12.794	74.41	1190.54	13.442
28	78.46	1255.3	12.744	77.42	1238.79	12.914
29	78.36	1253.83	12.759	73.73	1179.73	13.536
30	78.37	1253.98	12.761	75.44	1207.04	13.253
Gjennomsnitt	78.29333333	1252.702	12.7708	75.82166667	1213.137	13.192
avvik	0.16	2.49	0.03	1.22	19.51	0.21

Host mixed IO

kjøring	total mb/s	total iops	total latency
1	10.70	171.26	93.499
2	10.85	173.66	92.14
3	10.86	173.78	92.066
4	10.81	173.04	92.386
5	10.89	174.22	91.798
6	10.87	173.92	91.996
7	10.87	173.98	92
8	10.83	173.26	92.33
9	10.92	174.8	91.579
10	10.87	173.89	91.953
11	10.87	173.93	91.997
12	10.77	172.36	92.727
13	10.84	173.41	92.2
14	10.81	172.98	92.485
15	10.82	173.1	92.404
16	10.85	173.58	92.103
17	10.81	172.9	92.563
18	10.90	174.43	91.801
19	10.86	173.84	92.141
20	10.75	172.01	93.103
21	10.91	174.52	91.667
22	10.88	174.13	91.887
23	10.88	174.01	91.918
24	10.95	175.15	91.332
25	10.85	173.68	92.173
26	10.83	173.26	92.282
27	11.00	175.97	90.925
28	10.86	173.77	92.02
29	10.86	173.7	92.176
30	10.83	173.29	92.211
Gjennomsnitt	10.85333333	173.661	92.12873333
avvik	0.06	0.89	0.47

VM mixed IO

kjøring	total mb/s	total iops	total latency
1	10.41	166.59	96.034
2	10.44	167.08	95.763
3	10.46	167.39	95.613
4	10.43	166.8	95.935
5	10.45	167.26	95.658
6	10.40	166.34	96.147
7	10.50	167.97	95.269
8	10.45	167.22	95.611
9	10.50	168.02	95.226
10	10.45	167.2	95.688
11	10.45	167.24	95.653
12	10.46	167.35	95.599
13	10.45	167.26	95.65
14	10.45	167.25	95.634
15	10.43	166.94	95.781
16	10.46	167.38	95.535
17	10.45	167.27	95.707
18	10.48	167.66	95.421
19	10.37	165.9	96.487
20	10.44	166.97	95.81
21	10.46	167.42	95.534
22	10.46	167.4	95.601
23	10.48	167.67	95.497
24	10.42	166.64	96.028
25	10.46	167.41	95.55
26	10.44	167.05	95.776
27	10.44	167.01	95.81
28	10.48	167.65	95.447
29	10.43	166.83	95.896
30	10.45	167.27	95.59
Gjennomsnitt	10.44833333	167.1813333	95.69833333
avvik	0.03	0.44	0.25

Container mixed IO

kjøring	total mb/s	total iops	total latency
1	10.42	166.69	96.019
2	10.41	166.54	96.11
3	10.46	167.43	95.58
4	10.41	166.57	96.089
5	10.43	166.9	95.958
6	10.44	167.11	95.755
7	10.41	166.6	96.047
8	10.41	166.57	96.064
9	10.45	167.15	95.778
10	10.40	166.34	96.167
11	10.41	166.61	96.029
12	10.37	165.95	96.421
13	10.44	167.02	95.822
14	10.43	166.95	95.85
15	10.39	166.24	96.263
16	10.41	166.54	96.144
17	10.36	165.83	96.481
18	10.43	166.82	95.945
19	10.38	166.05	96.407
20	10.42	166.67	95.981
21	10.40	166.34	96.178
22	10.42	166.79	95.922
23	10.43	166.94	95.854
24	10.39	166.21	96.254
25	10.42	166.78	95.898
26	10.44	167.04	95.791
27	10.46	167.32	95.657
28	10.44	167.01	95.802
29	10.40	166.41	96.123
30	10.47	167.56	95.449
Gjennomsnitt	10.41833333	166.6993333	95.9946
avvik	0.03	0.42	0.24

TILLEGG

H

RÅDATA FRA TIDTAKING

	Oppstart		Stopp		
kjøring	Container	VM	Container	VM	
1	3.289551	18	1.5246715		2.6
2	3.6841208	18	2.3285792		2.7
3	3.3808863	17	1.9242157		2.8
4	3.6780808	17	1.6345062		2.4
5	3.5133606	17	1.9317884		2.7
6	3.5458952	17	2.0197971		2.7
7	3.6344655	17	1.4241895		2.6
8	3.5948735	17	1.4033091		2.5
9	3.2878971	17	2.0103776		2.6
10	3.3382367	17	1.6268031		2.5
11	3.4843002	19	1.4858244		2.7
12	3.5089277	18	1.407714		2.7
13	3.4358084	19	1.9736714		2.5
14	3.5605166	17	1.7847124		2.6
15	3.377411	18	2.0540574		2.5
16	3.5517264	17	1.7637863		2.5
17	3.3679029	17	1.6091525		2.6
18	3.7266043	18	1.8669645		2.7
19	3.5479292	18	1.9083836		2.5
20	3.7801955	17	1.6065141		2.7
21	3.7148177	20	1.6652974		2.6
22	3.4382346	17	2.0232421		2.7
23	3.4532318	17	1.4776748		2.5
24	3.3073045	17	1.879842		2.5
25	3.5871853	17	1.5422912		2.7
26	3.2608188	17	1.8763788		2.7
27	3.5643224	17	2.1329098		2.4
28	3.4839051	18	1.6702841		2.5
29	3.5425104	17	1.5211197		2.3
30	3.4052804	17	1.3641377		2.4
Gjennomsnitt	3.501543357	17.46666667	1.748073187		2.58
Avvik	0.14010141	0.776079152	0.251923646	0.121485064	

TILLEGG

I

EKSEMPELUTSKRIFT FRA EN KJØRING AV GEEKBENCH DIREKTE PÅ SERVER

```
System Information
  Operating System      Microsoft Windows Server 2016 Technical Preview 4 (64-bit)
  Model                 Dell Inc. PowerEdge T610
  Motherboard           Dell Inc. 09CGW2
  Processor             Intel Xeon E5506 @ 2.11 GHz
                        2 Processors, 8 Cores, 8 Threads
  Processor ID          GenuineIntel Family 6 Model 26 Stepping 5
  Processor Package     Socket 1366 LGA
  Processor Codename    Gainestown
  L1 Instruction Cache  32.0 KB x 4
  L1 Data Cache         32.0 KB x 4
  L2 Cache              256 KB x 4
  L3 Cache              4.00 MB
  Memory                24.0 GB DDR3 SDRAM -1MHz
  Northbridge           Intel 5520 13
  Southbridge           Intel 82801IB (ICH9) 02
  BIOS                  Dell Inc. 3.0.0
  Compiler              Visual C++ 170061030

Integer Performance
  AES
    single-core           145
    multi-core           1192 ||
  Twofish
    single-core          1568 |||
    multi-core          12525 |||||||||||||||||||||||||||
  SHA1
    single-core          2155 ||||
    multi-core          17278 |||||||||||||||||||||||||||||||||||||
  SHA2
    single-core          2309 |||||
    multi-core          18344 |||||||||||||||||||||||||||||||||||||||||
  BZip2 Compress
    single-core          1623 |||
    multi-core          12331 |||||||||||||||||||||||||
  BZip2 Decompress
    single-core          1503 |||
    multi-core          11355 |||||||||||||||||||||||
  JPEG Compress
    single-core          1753 |||
    multi-core          13966 |||||||||||||||||||||||||||||
  JPEG Decompress
    single-core          2350 |||||
    multi-core          13100 ||||||||||||||||||||||||||||
  PNG Compress
    single-core          1666 |||
    multi-core          13203 ||||||||||||||||||||||||||||
  PNG Decompress
    single-core          1664 |||
    multi-core          11384 |||||||||||||||||||||||||
  Sobel
    single-core          1626 |||
    multi-core          10290 ||||||||||||||||||||||
  Lua
    single-core          1597 |||
    multi-core          12279 ||||||||||||||||||||||||||
  Dijkstra
```

```
    single-core          1325 ||
    multi-core           5709 ||||||||||||

Floating Point Performance
  BlackScholes
    single-core          1947 ||||
    multi-core          15203 |||||||||||||||||||||||||||||||||
  Mandelbrot
    single-core          1799 |||
    multi-core          14099 |||||||||||||||||||||||||||||||
  Sharpen Filter
    single-core          1191 ||
    multi-core           8827 |||||||||||||||||||
  Blur Filter
    single-core          1193 ||
    multi-core           9516 ||||||||||||||||||||
  SGEMM
    single-core          1866 ||||
    multi-core          12510 ||||||||||||||||||||||||||||
  DGEMM
    single-core          1781 |||
    multi-core          12271 |||||||||||||||||||||||||||
  SFFT
    single-core          1461 |||
    multi-core          11418 |||||||||||||||||||||||||
  DFFT
    single-core          1327 ||
    multi-core          10405 ||||||||||||||||||||||||
  N-Body
    single-core          1689 |||
    multi-core          13279 ||||||||||||||||||||||||||||||
  Ray Trace
    single-core          1915 ||||
    multi-core          14866 |||||||||||||||||||||||||||||||||

Memory Performance
  Stream Copy
    single-core          1538 |||
    multi-core           3271 |||||||
  Stream Scale
    single-core          1460 |||
    multi-core           2235 ||||
  Stream Add
    single-core          1549 |||
    multi-core           2107 ||||
  Stream Triad
    single-core          1227 ||
    multi-core           2233 ||||

Benchmark Summary
  Integer Score              1434  10212
  Floating Point Score       1590  12055
  Memory Score               1437   2421

  Geekbench Score            1497   9391
```

TILLEGG

J

EKSEMPELUTSKRIFT FRA EN KJØRING AV DISKSPD PÅ LAPTOP

```
Command Line: C:\diskspd\amd64fre\diskspd.exe -c5G -d120 -r -w100 -t1 -o16 -b64k -h -L -Z1G
G:\testFile1.dat

Input parameters:

    timespan:   1
    -------------
    duration: 120s
    warm up time: 5s
    cool down time: 0s
    measuring latency
    random seed: 0
    path: 'G:\testFile1.dat'
        think time: 0ms
        burst size: 0
        software cache disabled
        hardware write cache disabled, writethrough on
        write buffer size: 1073741824
        performing write test
        block size: 65536
        using random I/O (alignment: 65536)
        number of outstanding I/O operations: 16
        thread stride size: 0
        threads per file: 1
        using I/O Completion Ports
        IO priority: normal



Results for timespan 1:
*******************************************************************************

actual test time:   120.02s
thread count:       1
proc count:     8

CPU |  Usage |  User  |  Kernel |  Idle
-------------------------------------------
   0|   3.92%|   0.16%|    3.76%|  96.08%
   1|   0.03%|   0.00%|    0.03%|  99.99%
   2|   0.56%|   0.00%|    0.56%|  99.44%
   3|   0.07%|   0.00%|    0.07%|  99.95%
   4|   0.25%|   0.08%|    0.17%|  99.75%
   5|   0.01%|   0.00%|    0.01%| 100.00%
   6|   0.03%|   0.00%|    0.03%|  99.97%
   7|   0.07%|   0.00%|    0.07%|  99.95%
-------------------------------------------
avg.|   0.62%|   0.03%|    0.59%|  99.39%

Total IO
thread |       bytes     |     I/Os     |     MB/s   |  I/O per s |  AvgLat  | LatStdDev |
file
-----------------------------------------------------------------------------------------------
-------
     0 |      1269432320 |        19370 |      10.09 |     161.40 |   99.125 |    35.262 |
     G:\testFile1.dat (5120MB)
```

```
-----------------------------------------------------------------------------------------------
-------
total:        1269432320 |        19370 |      10.09 |     161.40 |   99.125 |    35.262

Read IO
thread |       bytes     |     I/Os     |     MB/s   |  I/O per s |  AvgLat  | LatStdDev |
file
-----------------------------------------------------------------------------------------------
-------
     0 |               0 |            0 |       0.00 |       0.00 |    0.000 |       N/A |
     G:\testFile1.dat (5120MB)
-----------------------------------------------------------------------------------------------
-------
total:                 0 |            0 |       0.00 |       0.00 |    0.000 |       N/A

Write IO
thread |       bytes     |     I/Os     |     MB/s   |  I/O per s |  AvgLat  | LatStdDev |
file
-----------------------------------------------------------------------------------------------
-------
     0 |      1269432320 |        19370 |      10.09 |     161.40 |   99.125 |    35.262 |
     G:\testFile1.dat (5120MB)
-----------------------------------------------------------------------------------------------
-------
total:        1269432320 |        19370 |      10.09 |     161.40 |   99.125 |    35.262


  %-ile |  Read (ms) | Write (ms) | Total (ms)
-----------------------------------------------
    min |        N/A |      1.099 |      1.099
   25th |        N/A |     87.254 |     87.254
   50th |        N/A |     93.533 |     93.533
   75th |        N/A |    103.170 |    103.170
   90th |        N/A |    115.681 |    115.681
   95th |        N/A |    125.567 |    125.567
   99th |        N/A |    245.454 |    245.454
3-nines |        N/A |    559.678 |    559.678
4-nines |        N/A |    652.061 |    652.061
5-nines |        N/A |    655.042 |    655.042
6-nines |        N/A |    655.042 |    655.042
7-nines |        N/A |    655.042 |    655.042
8-nines |        N/A |    655.042 |    655.042
9-nines |        N/A |    655.042 |    655.042
    max |        N/A |    655.042 |    655.042
```

TILLEGG

K

EKSEMPELUTSKRIFT FRA EN KJØRING AV NTTTCP

```
<ntttcps computername="WIN-VA718QLRIN6" version="5.31">
    <parameters>
        <send_socket_buff>0</send_socket_buff>
        <recv_socket_buff>-1</recv_socket_buff>
        <port>5009</port>
        <sync_port>False</sync_port>
        <async>True</async>
        <verbose>False</verbose>
        <wsa>False</wsa>
        <use_ipv6>False</use_ipv6>
        <udp>False</udp>
        <verify_data>False</verify_data>
        <wait_all>False</wait_all>
        <run_time>15000</run_time>
        <warmup_time>7500</warmup_time>
        <cooldown_time>7500</cooldown_time>
        <dash_n_timeout>10800000</dash_n_timeout>
        <bind_sender>False</bind_sender>
        <sender_name></sender_name>
        <max_active_threads>8</max_active_threads>
    </parameters>
    <thread index="0">
        <realtime metric="s">15.032</realtime>
        <throughput metric="KB/s">11683.587</throughput>
        <throughput metric="MB/s">11.410</throughput>
        <throughput metric="mbps">95.712</throughput>
        <avg_bytes_per_compl metric="B">131072.000</avg_bytes_per_compl>
    </thread>
    <thread index="1">
        <realtime metric="s">15.032</realtime>
        <throughput metric="KB/s">12279.689</throughput>
        <throughput metric="MB/s">11.992</throughput>
        <throughput metric="mbps">100.595</throughput>
        <avg_bytes_per_compl metric="B">131072.000</avg_bytes_per_compl>
    </thread>
    <thread index="2">
        <realtime metric="s">15.032</realtime>
        <throughput metric="KB/s">11939.059</throughput>
        <throughput metric="MB/s">11.659</throughput>
        <throughput metric="mbps">97.805</throughput>
        <avg_bytes_per_compl metric="B">131072.000</avg_bytes_per_compl>
    </thread>
    <thread index="3">
        <realtime metric="s">15.032</realtime>
        <throughput metric="KB/s">12092.342</throughput>
        <throughput metric="MB/s">11.809</throughput>
        <throughput metric="mbps">99.060</throughput>
        <avg_bytes_per_compl metric="B">131072.000</avg_bytes_per_compl>
    </thread>
    <thread index="4">
        <realtime metric="s">15.032</realtime>
        <throughput metric="KB/s">11070.454</throughput>
        <throughput metric="MB/s">10.811</throughput>
        <throughput metric="mbps">90.689</throughput>
        <avg_bytes_per_compl metric="B">131072.000</avg_bytes_per_compl>
    </thread>
    <thread index="5">
```

```
        <realtime metric="s">15.032</realtime>
        <throughput metric="KB/s">12696.960</throughput>
        <throughput metric="MB/s">12.399</throughput>
        <throughput metric="mbps">104.013</throughput>
        <avg_bytes_per_compl metric="B">131072.000</avg_bytes_per_compl>
    </thread>
    <thread index="6">
        <realtime metric="s">15.032</realtime>
        <throughput metric="KB/s">11377.021</throughput>
        <throughput metric="MB/s">11.110</throughput>
        <throughput metric="mbps">93.201</throughput>
        <avg_bytes_per_compl metric="B">131072.000</avg_bytes_per_compl>
    </thread>
    <thread index="7">
        <realtime metric="s">15.032</realtime>
        <throughput metric="KB/s">10993.813</throughput>
        <throughput metric="MB/s">10.736</throughput>
        <throughput metric="mbps">90.061</throughput>
        <avg_bytes_per_compl metric="B">131072.000</avg_bytes_per_compl>
    </thread>
    <total_bytes metric="MB">1381.750000</total_bytes>
    <realtime metric="s">15.031000</realtime>
    <avg_bytes_per_compl metric="B">131072.000</avg_bytes_per_compl>
    <threads_avg_bytes_per_compl metric="B">131072.000</threads_avg_bytes_per_compl>
    <avg_frame_size metric="B">1459.491</avg_frame_size>
    <throughput metric="MB/s">91.927</throughput>
    <throughput metric="mbps">771.137</throughput>
    <total_buffers>11054.000</total_buffers>
    <throughput metric="buffers/s">735.413</throughput>
    <avg_packets_per_interrupt metric="packets/interrupt">0.701</avg_packets_per_interrupt>
    <interrupts metric="count/sec">29824.696</interrupts>
    <dpcs metric="count/sec">16561.706</dpcs>
    <avg_packets_per_dpc metric="packets/dpc">1.262</avg_packets_per_dpc>
    <cycles metric="cycles/byte">50.901</cycles>
    <packets_sent>992723</packets_sent>
    <packets_received>314265</packets_received>
    <packets_retransmitted>7892</packets_retransmitted>
    <errors>0</errors>
    <cpu metric="%">28.821</cpu>
    <bufferCount>9223372036854775807</bufferCount>
    <bufferLen>131072</bufferLen>
    <io>2</io>
</ntttcps>
```