|  |
| --- |
| Rapport |

Containrar och virtuella datorer

|  |
| --- |
| *Författare:* Samuel Berg  *Termin:* Ht 23  *Kursnamn:* Teknisk kommunikation  *Kurskod:* 1ZT010 |



Sammandrag

Syftet med denna litteraturstudie är att få en bättre förståelse för vad olika virtuella tekniker har att ge och undersöka eventuella säkerhetsrisker samt eventuella för- och nackdelar med respektive teknologi. Alla källor som använts i denna litteraturstudie är vetenskapligt granskade. En fördel för containrar är att de är portabla. En fördel för virtuella datorer (VMs) är att de är mer isolerade från dess värd-OS. I rapporten redovisas att det finns ingen självklart situation som kräver containrar eller VMs. Det som avgör är ändamålet som användaren strävar efter. Då det gäller tillkomna säkerhetsrisker så har containrar en större yta som kan bli attackerad i jämförelse med VMs.

The purpose of this literature study is to gain a better understanding of what different virtual technologies have to offer and to investigate possible security risks as well as possible pros and cons of each technology. All sources used in this literature study are scientifically reviewed. One advantage of containers is that they are portable. One advantage of virtual machines (VMs) is that they are more isolated from the host OS. The report states that there is no obvious situation that requires containers or VMs. The purpose that the user strives for determines the preferable technology to use. When it comes to added security risks, containers have a larger surface that can be attacked compared to VMs.

Nyckelord

Docker, Containrar, VMs, Virtuella datorer, Virtuella maskiner

Innehåll

1 Inledning I

1.1 Syfte och frågeställningar I

1.2 Metod och material I

2 Resultat II

2.1 Vilka för- och nackdelar finns vid användning av containrar respektive VMs? II

2.2 Vilka säkerhetsrisker har tillkommit? III

3 Diskussion och slutsatser IV

3.1 Diskussion IV

3.2 Slutsatser IV

Referenser V

# Inledning

Mjukvaruutvecklare är intresserade av att förbättra och utveckla både befintliga och nya verktyg, för att öka produktiviteten samt förenkla arbetssituationen vid de olika programmeringsprocedurerna. Denna litteraturstudie kommer att undersöka påverkan av *containers* (containrar) inom branschen för mjukvaruutveckling. I jämförelse med virtuella datorer (VMs). Då man arbetar vid en lokal enhet, exempelvis en dator, använder enheten generellt endast ett operativsystem vilket i många fall kan vara begränsande vid programmering. Därför har virtuell teknik utvecklats som ger användare möjlighet att samtidigt köra flera olika operativsystem och applikationer på en enda fysisk enhet. Arbetet kan då köras på det så kallade gäst-operativsystemet (gäst-OS) i stället för värd-operativsystemet (värd-OS). Värd-OS är det operativsystem (OS) som den fysiska enheten använder. I rapporten nämns även *images* vilket i detta sammanhang syftar på en avbild av allt från en applikation till ett OS.

Inom mjukvaruteknik finns det flera olika virtualiseringsmetoder som delas in flera olika klasser. I denna litteraturstudie studeras två olika klasser. Den första klassen är traditionell virtualisering, där en komplett virtuell maskin används. Det innebär traditionell virtualisering med en komplett hårdvaruemulering och ovanpå detta finns ett komplett operativsystem installerat. Den andra klassen kallas containerisering eller ofta *Lightweight-Virtualization* [1]*.*

## Syfte och frågeställningar

I denna litteraturstudie kommer två olika typer av virtuell teknik att jämföras, containrar och VMs. Dessa två system har valts på grund av att de i nuläget är de mest välkända och mest använda. Syftet med studien är att få en bättre förståelse för vad olika virtuella tekniker har att ge och undersöka eventuella säkerhetsrisker. Det är även önskvärt att kunna undersöka skillnaderna mellan de två systemen eftersom containrar och VMs blir mer och mer populära. Det blir då relevant att få reda på för- och nackdelar med respektive system. Denna litteraturstudie kommer att fokusera på följande frågeställningar:

FS1: Vilka för- och nackdelar finns vid användning av containrar respektive VMs inom mjukvaruutveckling?

FS2: Vilka säkerhetsrisker har tillkommit?

## Metod och material

Denna rapport är en litteraturstudie. Materialet utgörs av följande akademiska artiklar [1–8]. Dessa artiklar har hittats genom sökningar inom relevanta databaser för akademiska skrifter och genom att söka på relevanta ämnen inom dessa databaser. Artiklarna har valts då de är tydliga i sitt framförande av relevant information. Det kan också påpekas att de artiklarna som använts ger ut information som stämmer överens med de grundkunskaper jag själv har inom ämnet och utvidgar dessa. Artiklarna är i majoritet från olika konferenser vilket i mina ögon ger tillsammans en tydlig helhets bild på området och inte går in med en förutfattad tanke.

# Resultat

I den första delen av resultatkapitlet redovisas svar på den första frågeställningen, det vill säga för- och nackdelar vid användning av containrar och VMs. Här ges exempel på fördelar för containrar, exempelvis portabiliteten, skalbarheten samt resurseffektivitet.

En fördel med VMs är att systemet är mer isolerat. Några andra fördelar är möjligheten att köra flera olika OS samt att det är en äldre teknologi än containrar. I den andra resultatdelen presenteras de säkerhetsrisker som tillkommit med containrar.

## Vilka för- och nackdelar finns vid användning av containrar respektive VMs?

Mjukvaror som *Docker* gör det möjligt för ett enstaka värd-OS att isolera flera applikationer eller gäst-OS, i likhet med VMs. Detta genomförs inom en abstraktion som kallas containrar. Mjukvaruutvecklare som är oroliga för höga *input/output* (I/O) kostnader skulle kunna överväga *baremetal*-distributioner framför Docker-containerdistributioner. Baremetal-distributioner är en typ av OS som innehåller så få delar som möjligt. Det innebär att ett sådant system består av minimalt antal byggstenar för att fungera. I dessa sammanhang är det viktigt med egenskaper kopplade till driftsättning som portabilitet, skalbarhet och balans mellan mjukvarukrav och hårdvarukapacitet. Applikationer eller gäst-OS som används existerar dock inte på egen hand utan lagras på VMs eller containrar. Den mest populära applikationen för containerhantering som används idag är Docker [2], [6], [7], [8].

Docker byggs vanligtvis med Linux-baserade OS. Den modell som används är vanligtvis paketeringsmodellen och den utgår från respektive Linux-distribution. Detta innebär att det mesta av programvaran installeras som ett paket. När en image har byggts upp förblir paketen låsta för den version av paketen som imagen byggdes upp med. Majoriteten av paketen som ingår i imagen underhålls inte kontinuerligt utan de kan vara kvar i en version i flera månader innan en nyare version släpps. Först när en ny version av paketet har släppts kan en ny imageversion byggas upp. Den gamla versionen av imagen kan fortfarande vara i bruk eftersom den är utplacerad som en container i produktion. De containrar som motsvarar gamla images kan innehålla föråldrade paket med kända säkerhetsbrister och buggar. Eftersom containrarna kan köras i produktion kan de utsättas för personer med skadligt uppsåt. Varje ny version av images är alltså ofta en förbättrad version där buggar och säkerhetsrisker är borttagna [2], [3], [4], [8].

Då VMs och/eller containrar implementeras används beräkningsresurser på ett effektivt sätt. Processen att containerisera en applikation har ökat i popularitet på grund av att den lämnar ett mindre fotavtryck på hårdvaran av den fysiska enheten jämfört med VMs. En nyckelfunktion hos containrar är möjligheten att i produktion kunna migrera containrar tillsammans med de applikationer som stöds till en distinkt server eller enhet utan att orsaka tjänstavbrott [1], [2], [3], [4], [8].

Containerisering är ett tillvägagångssätt för mjukvaruutveckling som syftar till att paketera en applikation tillsammans med alla dess beroenden. Samtidigt byggs en exekveringsmiljö för att köra applikationen i en lätt, fristående enhet. I denna fråga har Docker blivit den ledande industristandarden. Genom att definiera den för Docker specifika image-arkitekturen och ordningen i byggandet spelar *dockerfile* en viktig roll i den Docker-baserade processen av uppbyggandet av containrar. Att förstå utvecklingen av *dockerfiles* och vilka byggnadstyper av dockerfiler som förbättrar dockerfilens kvalitet och minskar imagebyggandets tidsåtgång kan gynna den effektiva bearbetningen av containerisering [5].

För VMs är applikationerna och funktionaliteterna mer isolerande men hårdvarukrävande då det skapas ett nytt gäst-OS på den fysiska enheten. Det gäst-OS som skapats måste tilldelas resurser från enheten manuellt. Detta leder till att applikationen som annars hade kört via containrar i stället körs på ett gäst-OS. Detta gäst-OS gör miljön mer isolerad vilket betyder att om något problem uppstår med applikationen så påverkar det inte värd-OS utan endast det gäst-OS som applikationen körs på. Om samma applikation hade körts med en container hade även värd-OS påverkats i stället för att det skulle ha varit helt isolerat till gäst-OS. Ett värd-OS hade kunnat bli påverkat om containern skulle bli skadad. En sådan skada hade kunnat ge en permanent skada på värd-OS [1-3].

## Vilka säkerhetsrisker har tillkommit?

Docker är en teknologi som dominerar inom mjukvaruutveckling när det gäller hantering av containrar. Med dess ökande popularitet uppstår nya metoder som gör det möjligt att implementera nya säkerhetsmekanismer. Det dyker upp många olika lösningar som hanterar säkerhet på nivån för containerns imageskapande parallellt med teknologier som hanterar mjukvara och Docker-miljön [2], [4], [8].

Trots eventuella säkerhetsbrister och buggar i äldre versioner kan de som använder containrar föredra att hålla sig till äldre versioner av paket. Dessa är kända för att fungera bra och de har testats i produktion under längre tid. Faktum är att reproducerbarhet är en av de viktigaste egenskaperna hos Docker-containrar och även för utvecklare. Orsak till detta är att användning av *containerimages* tar bort de krav som alltid finns på det fysiska systemet, vilket innebär att containerimages ger isolering från förändringar som kan bryta fungerande system. En nyare version kan innebära att vissa funktionaliteter slutar fungera. Detta är ett starkt incitament för att hålla sig till en föråldrad image eftersom den huvudsakligen endast fungerar för den applicering den används i. En uppgradering till nya versioner av containerimage innebär alltid en viss risk. Sålunda balanserar användaren alltid mellan sitt behov av uppdatering till nya images och risken för sårbarhet och buggar. Även risken för att ett fungerande system går sönder på grund av oväntade ändringar i de uppgraderade paketen är en aspekt som användaren måste ta hänsyn till [2], [3], [4], [8].

Som nämndes under föregående underrubrik finns det för- och nackdelar med containrar, det finns säkerhetsbrister och buggar i versioner av paket som inte underhålls på heltid. **Detta problem är mer vanligt för containrar eftersom den version som applikationerna körs i är mindre underhållen och det är svårare att byta än i VMs** [2], [3], [4], [8].

# Diskussion och slutsatser

I detta kapitel hålls först en kort diskussion om vad som undersökts i litteraturstudien med en sammanfattning samt en tolkning av resultaten. I den andra delen dras slutsatser och görs tolkningar från alla de resultat som nämnts i rapporten. I den andra delen nämns även ett annat alternativ för mjukvaruutveckling. Alternativet ingår inte i denna studie men är en möjlig fortsättning av denna studie.

## Diskussion

I denna litteraturstudie har jag undersökt för- och nackdelar med VMs och containrar samt sökt efter tillkomna säkerhetsrisker för containrar som är en nyare typ av teknologi än VMs. De för- och nackdelar som framkommit innebär att det finns tillfällen då det är mer lämpligt att använda den ena teknologin än den andra. Även de säkerhetsrisker som framkommit i undersökningen pekar mot att VMs och containrar lämpar sig olika bra för olika tillfällen. VMs är den lämpligare teknologin att använda när det är viktigt att hålla mjukvaran isolerad från värd-OS. Ett annat exempel är att det finns fler resurser som kan hjälpa användare då man stöter på problem. En annan anledning till att man väljer VMs över containrar handlar om att VMs inte är beroende av användarens fysiska enhets värd-OS. Med detta syftar jag på att en container alltid körs på samma OS som de värd-OS den ligger på, vilket kan leda till nackdelar då man vill köra mjukvaran på ett specifikt OS som inte är det samma som värd-OS. Det finns situationer då man inte bör använda ett VM, exempelvis då mjukvaran är väldigt resurseffektiv samt då man önskar en snabb uppstartstid. En snabb uppstartstid är något som börjar bli ett mer vanligt förekommande krav för applikationer och system i dagens värld.

Då användaren har behov av en portabel mjukvara som kan köras överallt utan några större krav så är containrar mer lämpliga att använda. Containrar föredras även då man har en fysisk enhet med en mindre mängd resurser som kan tilldelas vid användning för virtualiseringen. Detta leder till att man kan köra flera containrar på samma mängd resurser som endast en VM instans. I jämförelse med VMs så har containrar en väldigt bra skalbarhet då det är enkelt att bygga upp en ny container med de senaste applikationsversionerna. För samma förändring kräver ett VM en total ominstallering inklusive all mjukvara. Då det finns högre krav på säkerheten för mjukvaran inom vitaliseringen så är det mer lämpligt att använda VMs. Även då användaren behöver ett enklare sätt att koppla upp instansen till internet så är VMs att föredra.

Det finns också en del kända för- och nackdelar som inte har tagits upp i denna litteraturstudie då det inte funnits någon relevant studie som har pekar just på dessa. Möjligheten att kombinera dessa två virtualiseringsmetoder skulle kunna vara en möjlig fortsättning av denna undersökning.

## Slutsatser

Denna litteraturstudie har visat att det finns flertal för- och nackdelar med VMs respektive containrar vilket ger en tydligare bild av när och i vilken situation den ena respektive den andra teknologin ska användas. Båda teknologierna kan oftast användas men det kan finnas tillfällen då det är mer lämpligt att använda den ena över den andra. Containrar är att föredra då användaren önskar något som är portabelt samt är resurseffektivt. VMs är att föredra då det önskas en mer isolerad miljö för mjukvaran som körs samt om säkerhetsriskerna ska minimeras. Containrar har fler säkerhetsrisker än VMs eftersom det är en nyare teknologi, två exempel är att containrar dels ger en ökad attackyta, dels att det finns större risker med äldre versioner av paket och ramverk.

# Referenser

[1] A. Abuabdo and Z. A. Al-Sharif, "Virtualization vs. Containerization: Towards a Multithreaded Performance Evaluation Approach," 2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA), Abu Dhabi, United Arab Emirates, 2019, pp. 1-6, doi: [10.1109/AICCSA47632.2019.9035233](https://doi.org/10.1109/AICCSA47632.2019.9035233).

[2] D. Ukene, H. Wimmer and J. Kim, "Evaluating the Performance of Containerized Webservers against web servers on Virtual Machines using Bombardment and Siege," 2023 IEEE/ACIS 21st International Conference on Software Engineering Research, Management and Applications (SERA), Orlando, FL, USA, 2023, pp. 144-152, doi: [10.1109/SERA57763.2023.10197818](https://doi.org/10.1109/SERA57763.2023.10197818).

[3] S. Ramanathan, K. Kondepu, M. Tacca, L. Valcarenghi, M. Razo and A. Fumagalli, "Container Migration of Core Network Component in Cloud-Native Radio Access Network," 2020 22nd International Conference on Transparent Optical Networks (ICTON), Bari, Italy, 2020, pp. 1-5, doi: [10.1109/ICTON51198.2020.9328898](https://doi.org/10.1109/ICTON51198.2020.9328898).

[4] A. Maruszczak, M. Walkowski and S. Sujecki, "Base Systems for Docker Containers - Security Analysis," 2022 International Conference on Software, Telecommunications and Computer Networks (SoftCOM), Split, Croatia, 2022, pp. 1-5, doi: [10.23919/SoftCOM55329.2022.9911523](https://doi.org/10.23919/SoftCOM55329.2022.9911523).

[5] Y. Zhang, G. Yin, T. Wang, Y. Yu and H. Wang, "An Insight Into the Impact of Dockerfile Evolutionary Trajectories on Quality and Latency," 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), Tokyo, Japan, 2018, pp. 138-143, doi: [10.1109/COMPSAC.2018.00026](https://doi.org/10.1109/COMPSAC.2018.00026).

[6] E. A. Santos, C. McLean, C. Solinas, and A. Hindle, "How does Docker affect energy consumption? Evaluating workloads in and out of Docker containers," The Journal of Systems and Software, vol. 146, pp. 14–25, 2018. doi: [10.1016/j.jss.2018.07.077](https://doi.org/10.1016/j.jss.2018.07.077).

[7] R. Kałaska and P. Czarnul, "Investigation of Performance and Configuration of a Selected IoT System—Middleware Deployment Benchmarking and Recommendations," Applied Sciences, vol. 12, no. 10, p. 5212, 2022, doi: [10.3390/app12105212](https://doi.org/10.3390/app12105212).

[8] A. Zerouali, T. Mens, G. Robles and J. M. Gonzalez-Barahona, "On the Relation between Outdated Docker Containers, Severity Vulnerabilities, and Bugs," 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), Hangzhou, China, 2019, pp. 491-501, doi: [10.1109/SANER.2019.8668013](https://doi.org/10.1109/SANER.2019.8668013).