|  |
| --- |
| Rapport |

Containrar och virtuella datorer

*En jämförlse om användar område*

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



Sammandrag

Skriv in abstrakt (ca 200 ord)

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 med containrar jämfört med VMs? 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 hos de olika procedurerna vid programmering.

Denna litteraturstudie kommer att undersöka påverkan av *Docker* och *containers* (containrar) inom mjukvaruutvecklings branschen jämfört 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 användaren 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.

En av klassifikationerna delar in dessa tekniker i två klasser. För det första, traditionell virtualisering, där en komplett virtuell maskin används, använder denna klass en komplett hårdvaruemulering, ovanpå detta ett komplett operativsystem installerat. Den andra klassen kallas *containerization* (containerisera) 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 de i nuläget är de av allmänheten mest använda teknologierna inom programmering. 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 de skillnaderna mellan de två systemen eftersom containrar och VMs är något som blir mer och mer populärt. Det blir då relevant att upptäcka fördelar och nackdelar med respektive system. Denna litteraturstudie kommer att fokusera på följande frågeställningar:

FS1: Vilka fördelar och nackdelar finns vid användning av containrar respektive virtuella datorer (VMs) inom mjukvaruutveckling.

FS2: Vilka säkerhetsrisker har tillkommit med containrar jämfört med VMs.

## Metod och material

Denna rapport är en litteraturstudie. Materialet utgörs av följande akademiska artiklar [1–8].

# Resultat

I den första delen av resultatet redovisas de svar som hittats angående den första frågeställning, det vill säga de för- och nackdelar som finns vid användning av containrar och VMs, exempel på några fördelar för containers är portabiliteten, skalbarheten samt resurseffektiviteten. För att också nämna några av fördelarna för VMs isolerat, möjlighet till flera olika OS samt mognaden av att det är en äldre teknologi än containers. 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, liknande virtuella datorer. Detta görs inom en abstrakt abstraktion som kallas containrar. I samtliga fall var det en statistiskt signifikant ökning av energiförbrukningen när man körde tester i Docker, mestadels på grund av prestanda för input/output (I/O) systemanrop. Utvecklare som är oroliga för höga I/O kostnader skulle kunna överväga *baremetal*-distributioner framför Docker-containerdistributioner. Baremetal-distributioner vilket innefattar distributioner som är precis tillräckligt för att vara ett OS. I dessa sammanhang är det viktigt med egenskaper kopplade till driftsättning som portabilitet, skalbarhet och balans mellan mjukvarukrav och hårdvarukapacitet. Dessa applikationer eller gäst-OS:en som används existerar dock inte på egen hand, utan lagras på VMs eller containrar. Den mest populära container hanterings applikationen 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 ifrån respektive Linux-distribution. Detta innebär att det mesta av programvaran installeras som ett paket. När *image*:en har byggts upp så förblir paketen låsta för den versionen av image:en som har byggts upp. Majoriteten av paketen som ingår i image:en är inte heltids underhållna utan att de kan vara kvar i en version i flera månader innan en nyare version släpps. Först när en ny version av paketet har släppts så kan en ny imageversion byggas upp. Den gamla versionen av image:en kan fortfarande vara i bruk eftersom den är utplacerad som en container i produktion. De containrar som motsvarar gamla image:s kan innehålla föråldrade paket med kända säkerhetsbrister och buggar. Eftersom behållarna kan köras i produktion kan de utsättas för utnyttjande av personer med skadligt uppsåt. Varje ny version av images är alltså ofta en förbättrad version där dessa buggar och säkerhetsrisker är borttagna [2], [3], [4], [8].

Trots eventuella säkerhetsbrister och buggar i äldre versioner så finns det möjlighet att de som använder containrar föredrar att hålla sig till äldre versioner av paket. Detta beror på paketen 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 för utvecklare. Orsak till detta är att användning av containerimages ger isolering från utvecklande beroenden och förändringar i paket som kan bryta fungerande system. Det betyder kort sagt 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 "bara fungerar" för den applicering den används i. En uppgradering till nya versioner av containerimage innebär alltid en viss risk. Sålunda balanserar arbetsgivare alltid mellan sitt behov av uppdatering till nya images och den möjliga sårbarheten och buggar samt risk för att ett fungerande system går sönder på grund av oväntade ändringar i de uppgraderade paketen [2], [3], [4], [8].

Då VMs eller/och containrar implementeras så används beräkningsresurser på ett resurseffektivt sätt. Processen att containerisera en applikation har ökat i popularitet på grund av att det lämnar ett mindre fotavtryck på hårdvaran av den fysiska enheten jämfört med VMs. En nyckelfunktion av containers är möjligheten att i produktion kunna migrera containrar tillsammans med deras 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 som behövs för att köra applikationen i en lätt, fristående enhet. I denna fråga har Docker blivit den ledande industristandarden. Genom att definiera den specifika Docker image-arkitekturen och ordningen i byggandet spelar *dockerfile* en viktig roll i den Docker-baserade processen av uppbyggandet av containers. 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].

När det kommer till VMs så är applikationerna och funktionaliteterna mer isolerande men hårdvarukrävande då det skapas ett nytt gäst-OS på din fysiska enhet. För det gäst-OS som skapats måste det tilldelas resurser från din enhet manuellt. Detta leder till att applikationen som du annars hade kört via containrar körs i stället på ett gäst-OS som gör det mer isolerat vilket kort förklarat betyder att om något problem uppstår med applikationen så påverkar det inte ditt värd-OS utan endast det gäst-OS:et som applikationen körs på. Om man jämför det med vad som hade hänt om samma applikation hade körts med en container så hade även ditt värd-OS påverkats på något sätt i stället för att det skulle ha varit helt isolerat till gäst-OS:et [1-3].

## Vilka säkerhetsrisker har tillkommit med containrar jämfört med VMs?

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].

Som nämndes under föregående underrubrik så finns det för oc nackdelar med containrar när det kommer till att säkerhetsbrister och buggar som finns i versioner av paket som inte underhålls på heltid vilket är ett mer vanligt problem jämfört med underhållningen av hur den typen av applikationer körs i VMs där versionerna ofta är mer underhållna [2], [3], [4], [8].

# Diskussion och slutsatser

Metatext

## Diskussion

Diskutera

## Slutsatser

Dra slutsatser

# 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.

[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.

[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.

[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.

[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.

[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.

[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.

[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.