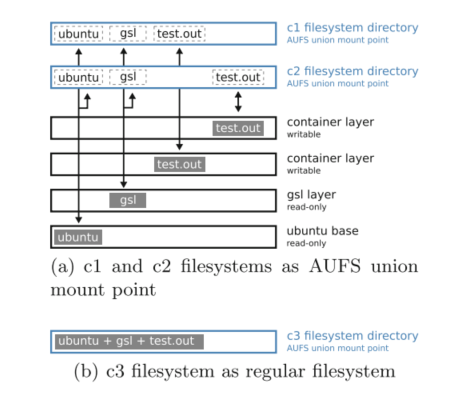
Experiment – Reproductie

# Inleiding

Het doel van dit experiment is de reproductie van de resultaten die bereikt zijn in het artikel “More sharing, more benefits? A study of library sharing”. [1]

In dit artikel wordt er in het motivatie onderdeel wat resultaten van een klein experiment voorgesteld. Deze tonen aan de libraries in geheugen gedeeld worden via AUFS. Dit bestandssysteem wordt ook gebruikt in Docker en momenteel wordt zelfs de nieuwere overlay2 vooral gebruikt. Het experiment wordt zo opgesteld: Er zijn 3 containers genaamd C1, C2 en C3 voor hen geldt dat C1 en C2 hun bestandssystemen AUFS union mount points zijn bestaande uit 3 verschillende lagen. Deze zijnde Ubuntu base layer, GSL (GNU scientific libraries) library layer en de container layer met een applicatie die gebruik maakt van GSL. C3 heeft dezelfde componenten maar zal deze op ext4 bestandssysteem hebben staan. De onderstaande afbeelding uit het artikel geeft de situatie weer.

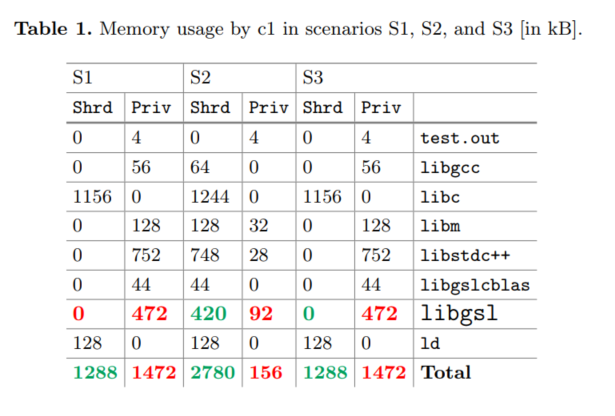


Figuur : Opstelling van de container lagen

In het experiment worden deze containers a.d.h.v. de container technologie LXC gemaakt. [2] Verder worden er 3 scenario’s opgesteld om metingen op uit te voeren.

* Enkel C1 draait op de host
  + Verwacht resultaat weinig of geen sharing want er is maar 1 container
* C1 en C2 draaien op de host
  + Verwacht resultaat is hoge sharing tussen beide containers
* C1 en C3 draaien op de host
  + Weinig tot geen sharing tussen de containers

Om de metingen uit te voeren wordt er gebruik gemaakt van het commando pmap [3], die de memory map van een proces weergeeft. In de onderstaande afbeelding worden hun resultaten weergegeven en deze duiden dan uiteindelijk aan dat sharing wel degelijk gebeurd door de AUFS lagen. Hieruit berekenen ze dan voor de verschillende scenario’s de totale kost en hieruit blijkt dat in S2 de kost = 2780 + 2 \* 156 = 3092kB en voor S3 is dit = 1288 + 2 \* 1472 = 4232 kB.



Dit was een inleidend onderdelen naar het origineel experiment om hun resultaten en methodiek aan te duiden. In de rest van de tekst gaan we eerst kijken naar de problemen in hun methodiek en verder worden er zelf experimenten uitgevoerd om deze resultaten te verifiëren.

# Problemen

Na analyse van het experiment en wat planning omtrent de opzet ervan kwamen er enkele grote problemen tevoorschijn:

* Geen uitleg over de applicatie en hoe deze is opgebouwd
  + We maken hier logische assumpties maar kan de reproductie gedeeltelijk in gedrang brengen.
* De omzetting van de data die pmap geeft en de tabel wordt niet benadrukt
  + Dit geeft initieel wat moeilijkheden
* Er wordt geen informatie gegeven over de LXC container geconfigureerd worden en hoe de AUFS mount point ingesteld worden.
  + Dit vroeg over redelijk wat assumpties en na opzoek werk omtrent AUFS en LXC leek mij ook hun opgezette situatie niet overeen te komen met mijn idee ervan.

# Experimenten

## Experiment 1: Reproductie

### Inleiding

Initieel wou ik een letterlijke reproductie van hun setup proberen te realiseren. Maar tijdens het proberen nabootsen van deze setup kwam ik meer en meer op vraagtekens uit omtrent hun setup en dit bracht volgens mij de waarde van de reproductie in gedrang.

Hierdoor heb ik het over een andere boeg gegooid en ga ik wel gebruik maken van hun manier om shared libraries te meten a.d.h.v. tools zoals pmap [3] maar dan in een Docker omgeving die gebruik maakt van een andere UnionFS (overlay2), deze is een verbeterde versie van AUFS. Verder zal er ook gebruik gemaakt worden van /proc/<pid>/smaps [4] omdat deze meer informatie bevat.

Overlay2 is een verbeterde versie van het AUFS systeem en momenteel wordt dit standaard gebruikt voor Docker dus lijkt het mij relevant dat als we dit proberen te verifiëren dat dit ook gebeurd op een platform dat momenteel waarde heeft. [5]

Verder wordt hun opstelling ook gevolgd in termen van scenario’s buiten dat we nu maar 2 scenario’s opstellen. Scenario 1 is het gebruik van 1 Pod die de baseline aangeeft van het gebruik. Het tweede scenario duidt het gebruik van gedeelde libraries aan door dezelfde Pod een tweede maal op te starten. Als evaluatie wordt dan de kost berekent van 1 container op 2 nodes te draaien en 2 containers op 1 node.

Wat we als laatste ook overnemen is de opbouw van het programma zoals in het originele experiment door van een Ubuntu basis te starten met de GSL layer hierboven en als laatste een executable “a.out” die gebruik maakt van de GSL libraries. Het compileren van deze applicatie werd gedaan zodat shared libraries gebruikt werden. [6]

#include <unistd.h>

#include <stdio.h>

#include <gsl/gsl\_math.h>

*int* main (*void*) {

    for ( ; ; ) {

*double* x = 5.0;

*double* y = gsl\_expm1(100.0);

    printf ("J0(%g) = %.18e\n", x, y);

    sleep(2);

    }

    return 0;

}

### Legende

Er zijn enkele benamingen die belangrijk zijn om te weten hoe de data gelezen moet worden. [7] [8] De meest relevante metrieken zijn shared en private omdat deze duidelijk aanduiden wat het verschil is tussen gedeeld geheugen en privaat geheugen.

* RSS = Resident Set Size
  + Totaal aantal geheugen in gebruik door een proces in RAM
  + RSS is redelijk onbelangrijk en geeft geen goed beeld op het delen van geheugen.
* PSS = Proportional Set Size
  + Gedeelte van RAM geheugen dat door een proces in gebruik genomen is en bestaat uit private geheugen + proportie van het gedeeld geheugen met één of meer processen.
  + = Unshared + gedeelte shared geheugen
* Shared clean en dirty
  + Clean betekend dat deze niet meer aangepast zijn sinds gemapped (tekst secties van shared libraries bv.) Dirty zijn pages die wel aangepast zijn.
  + Deze zullen meestal samengevoegd getoond worden in de gegevens
* Private clean en dirty
  + Hetzelfde geldt voor private en private memory is enkel beschikbaar voor dat proces terwijl shared met andere processen gedeeld wordt.
  + Deze zullen meestal samengevoegd getoond worden in de gegevens

### Metingen

Zoals eerder vermeld zal er gebruik gemaakt worden van dezelfde tools initieel zal pmap gebruikt worden om een vergelijking te maken maar nadien om een gedetailleerder beeld te krijgen zal er gekeken worden naar /proc/<pid>/smaps die veel meer informatie geeft over het geheugen gebruik.

De onderstaande output van pmap is al verwerkt en in tabelvorm gegoten voor duidelijkheid en de verschillen aan te tonen.

Tabel : pmap resultaten

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Procesnaam** | RSS(1) | PSS(1) | RSS(2) | PSS(2) | RSS(diff) | PSS(diff) |
| a.out | 12 | 12 | 12 | 10 | 0 | 2 |
| libm-2.19.so | 268 | 268 | 280 | 158 | -12 | 110 |
| libc-2.19.so | 980 | 552 | 1020 | 308 | -40 | 244 |
| libgslcblas.so.0.0.0 | 68 | 68 | 68 | 38 | 0 | 30 |
| libgsl.so.19.3.0 | 496 | 496 | 520 | 312 | -24 | 184 |
| ld-2.19.so | 136 | 72 | 212 | 99 | -76 | -27 |

De bovenstaande tabel toont in de eerste kolom de namen van de shared libraries. De rest van de tabel is opgedeeld in de 2 scenario’s en als laatste het verschil tussen de twee scenario’s. In het eerste scenario zien we dat enkel libc-2.19 en ld-2.19 hun RSS en PSS waardes niet identiek zijn. . Zoals eerder vermeld geeft PSS de proportionele set size en dit betekent dat libc-2.19 en ld-2.19 gedeeld al ergens anders in geheugen geladen zijn en gedeeld worden.

In het tweede scenario is het duidelijk dat de PSS vaak veel lager ligt dan de RSS, dit duidt dan op het delen van geheugen voor deze libraries. PSS is proportioneel dus dit houdt de waarde gedeeld over de processen die gebruik maken van de library.

Hier hebben we al een initieel beeld dat er duidelijk gedeeld wordt tussen containers. Om dit verder te ondersteunen is er ook gekeken naar /proc/<pid>/smaps die nog meer informatie geeft over de verschillende onderdelen. Deze geeft veel meer informatie maar de informatie in onderstaande tabellen is gecondenseerd en geeft het juiste beeld aan.

Tabel 2: /proc/<pid>/smaps: 1 container

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Proces** | **RSS** | **PSS** | **Shared** | **Private** |
| /a.out | 12 | 12 | 0 | 12 |
| /lib/x86\_64-linux-gnu/libm-2.19.so | 264 | 264 | 0 | 264 |
| /lib/x86\_64-linux-gnu/libc-2.19.so | 968 | 516 | 904 | 64 |
| /gnu/gsl/lib/libgslcblas.so.0.0.0 | 68 | 68 | 0 | 68 |
| /gnu/gsl/lib/libgsl.so.19.3.0 | 492 | 492 | 0 | 492 |
| /lib/x86\_64-linux-gnu/ld-2.19.so | 148 | 84 | 128 | 20 |
| **TOTAAL** | **1952** | **1436** | **1032** | **920** |

Tabel 3: /proc/<id>/smaps: 2 containers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Proces** | **RSS** | **PSS** | **Shared** | **Private** |
| /a.out | 12 | 10 | 4 | 8 |
| /lib/x86\_64-linux-gnu/libm-2.19.so | 252 | 140 | 224 | 28 |
| /lib/x86\_64-linux-gnu/libc-2.19.so | 948 | 265 | 924 | 24 |
| /gnu/gsl/lib/libgslcblas.so.0.0.0 | 68 | 38 | 60 | 8 |
| /gnu/gsl/lib/libgsl.so.19.3.0 | 480 | 282 | 396 | 84 |
| /lib/x86\_64-linux-gnu/ld-2.19.so | 148 | 43 | 140 | 8 |
| **TOTAAL** | **1908** | **778** | **1748** | **160** |

Uit de bovenstaande gegevens blijkt dus dat er gedeeld wordt en geeft een beter beeld. Om aan PSS te komen kan (Shared / #containers) + Private gedaan worden.

### Evaluatie

Om dit om te zetten naar bruikbare cijfers stellen we twee scenario’s voor (i) dus het 1 container model wordt op 2 nodes gedraaid en (ii) het 2 container model op 1 node. Dit geeft het voordeel van de layering in Docker weer op basis van kostenefficiëntie.

* Scenario 1: Container op 2 nodes (volgens 1 container tabel)
  + Totale kost = 2 \* (Shared + Private)
  + 2 \* (1032 + 920) = 3904
* Scenario 2: Container op 1 node (volgens 2 containers tabel)
  + Totale kost = Shared + 2 \* Private
  + 1748 + 2 \* 160 = 2068

Hieruit blijkt dus dat het enorm voordeliger is om de containers te draaien op dezelfde node als ze dezelfde image hebben of gedeelde layers.

## Experiment 2: NGINX

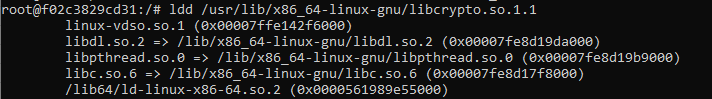
### Inleiding

In het bovenstaande experiment zijn de meeste concepten aangehaald en deze gaan hier gebruikt worden om te verifiëren dat dit ook geldt voor een NGINX image. Er wordt hier gebruik gemaakt van een basis NGINX docker image.

Net zoals voordien wordt dezelfde situatie opgezet met twee scenario’s om de vergelijking te duiden. Eerst het scenario met één container om de baseline te definiëren en dan twee containers om het verschil aan te duiden.

### Metingen

Onderstaande tabel toont het gebruik van één container en geeft aan dat i.v.m. met het vorige experiment er toch al veel gedeeld wordt standaard. Dit komt omdat veel van deze libraries bestaan uit andere shared libraries. Zoals in figuur 1 zichtbaar is voor de library libcrypto.



Figuur 2: libcrypto shared object dependencies

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Proces** | **RSS** | **PSS** | **Shared** | **Private** |
| /usr/sbin/nginx | 1116 | 864 | 504 | 612 |
| /lib/x86\_64-linux-gnu/libnss\_files-2.28.so | 56 | 28 | 56 | 0 |
| /lib/x86\_64-linux-gnu/libc-2.28.so | 1440 | 952 | 976 | 464 |
| /lib/x86\_64-linux-gnu/libz.so.1.2.11 | 72 | 68 | 8 | 64 |
| /usr/lib/x86\_64-linux-gnu/libcrypto.so.1.1 | 2140 | 2040 | 200 | 1940 |
| /usr/lib/x86\_64-linux-gnu/libssl.so.1.1 | 388 | 362 | 52 | 336 |
| /lib/x86\_64-linux-gnu/libpcre.so.3.13.3 | 80 | 76 | 8 | 72 |
| /lib/x86\_64-linux-gnu/libcrypt-2.28.so | 36 | 32 | 8 | 28 |
| /lib/x86\_64-linux-gnu/libpthread-2.28.so | 116 | 84 | 64 | 52 |
| /lib/x86\_64-linux-gnu/libdl-2.28.so | 16 | 12 | 8 | 8 |
| /lib/x86\_64-linux-gnu/ld-2.28.so | 164 | 102 | 124 | 40 |
| **TOTAAL** | **5624** | **4620** | **2008** | **3616** |

Tabel : /proc/<pid>/smaps van 1 NGINX

De volgende tabel voor twee containers maakt dan weer duidelijk dat sharing wel degelijk gebeurd op een hoger niveau.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Proces** | **RSS** | **PSS** | **Shared** | **Private** |
| /usr/sbin/nginx | 1140 | 492 | 1128 | 12 |
| /lib/x86\_64-linux-gnu/libnss\_files-2.28.so | 56 | 16 | 56 | 0 |
| /lib/x86\_64-linux-gnu/libc-2.28.so | 1464 | 523 | 1416 | 48 |
| /lib/x86\_64-linux-gnu/libz.so.1.2.11 | 72 | 36 | 72 | 0 |
| /usr/lib/x86\_64-linux-gnu/libcrypto.so.1.1 | 2248 | 1198 | 2100 | 148 |
| /usr/lib/x86\_64-linux-gnu/libssl.so.1.1 | 380 | 210 | 340 | 40 |
| /lib/x86\_64-linux-gnu/libpcre.so.3.13.3 | 80 | 40 | 80 | 0 |
| /lib/x86\_64-linux-gnu/libcrypt-2.28.so | 36 | 18 | 36 | 0 |
| /lib/x86\_64-linux-gnu/libpthread-2.28.so | 116 | 45 | 112 | 4 |
| /lib/x86\_64-linux-gnu/libdl-2.28.so | 16 | 8 | 16 | 0 |
| /lib/x86\_64-linux-gnu/ld-2.28.so | 164 | 46 | 160 | 4 |
| **TOTAAL** | **5772** | **2632** | **5516** | **256** |

Tabel : /proc/<pid>/smaps van 2 NGINX containers

Terwijl dit een beperkt experiment is wordt het toch meteen duidelijk dat de libraries wel degelijk gedeeld worden tussen containers.

### Evaluatie

Identiek aan het vorige experiment zal ik de berekeningen uitvoeren om aan te tonen dat dit voordeliger is.

* Scenario 1: NGINX container op 2 nodes (volgens 1 container tabel)
  + Totale kost = 2 \* (Shared + Private)
  + 2 \* (2008 + 3616) = 11248
* Scenario 2: NGINX containers op 1 node (volgens 2 containers tabel)
  + Totale kost = Shared + 2 \* Private
  + 5516 + 2 \* 256 = 6028

We komen hier weer tot dezelfde conclusie als voorheen.

## Experiment 3: Kubernetes + reproductie

Normaal gezien zou dit ook gewoon moeten transfereren naar Kubernetes omdat deze gebruik maakt van de Docker runtime. Er wordt toch echter geprobeerd de reproductie te runnen via een Deployment op Kubernetes om dit te verifiëren.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Proces | RSS | PSS | Shared | Private |
| /a.out | 12 | 12 | 0 | 12 |
| /lib/x86\_64-linux-gnu/libm-2.19.so | 524 | 524 | 0 | 524 |
| /lib/x86\_64-linux-gnu/libc-2.19.so | 1092 | 614 | 956 | 136 |
| /gnu/gsl/lib/libgslcblas.so.0.0.0 | 72 | 72 | 0 | 72 |
| /gnu/gsl/lib/libgsl.so.19.3.0 | 684 | 684 | 0 | 684 |
| /lib/x86\_64-linux-gnu/ld-2.19.so | 148 | 78 | 140 | 8 |
| **Totaal** | **2532** | **1984** | **1096** | **1436** |

Bovenstaande tabel is de tabel voor één container een geeft een kleine shared en grote private memory waarde weer.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Proces | RSS | PSS | Shared | Private |
| /a.out | 12 | 10 | 4 | 8 |
| /lib/x86\_64-linux-gnu/libm-2.19.so | 492 | 314 | 356 | 136 |
| /lib/x86\_64-linux-gnu/libc-2.19.so | 1124 | 370 | 1036 | 88 |
| /gnu/gsl/lib/libgslcblas.so.0.0.0 | 72 | 40 | 64 | 8 |
| /gnu/gsl/lib/libgsl.so.19.3.0 | 660 | 398 | 524 | 136 |
| /lib/x86\_64-linux-gnu/ld-2.19.so | 148 | 43 | 140 | 8 |
| **Totaal** | **2508** | **1175** | **2124** | **384** |

Deze tabel geeft vergelijkend met het eerste experiment een grotere shared omgeving weer en kleinere private omgeving en verifieert dus dat delen gebeurd.

## Experiment 4: Kubernetes + mt-api

Om te kijken naar onze applicatie zal ik het npm proces monitoren en zien hoeveel sharing hierin voorkomt.

|  |  |  |  |
| --- | --- | --- | --- |
| Procesnaam | RSS | Shared | Private |
| /usr/bin/node | 23556 | 23440 | 116 |
| /lib/x86\_64-linux-gnu/libc-2.28.so | 1480 | 1428 | 28 |
| /lib/x86\_64-linux-gnu/libpthread-2.28.so | 116 | 108 | 8 |
| /lib/x86\_64-linux-gnu/libgcc\_s.so.1 | 96 | 88 | 8 |
| /lib/x86\_64-linux-gnu/libm-2.28.so | 776 | 716 | 60 |
| /usr/lib/x86\_64-linux-gnu/libstdc++.so.6.0.25 | 1340 | 1240 | 100 |
| /lib/x86\_64-linux-gnu/librt-2.28.so | 32 | 24 | 8 |
| /lib/x86\_64-linux-gnu/libdl-2.28.so | 16 | 8 | 8 |
| /lib/x86\_64-linux-gnu/ld-2.28.so | 164 | 156 | 8 |
| **Totaal** | **27576** | **27208** | **344** |

Hier wordt al meteen duidelijk dat er standaard al enorm veel libraries gedeeld worden. Dit kan zijn omdat Kubernetes applicaties draait die hier ook gebruik van maken.

|  |  |  |  |
| --- | --- | --- | --- |
| Procesnaam | RSS | Shared | Private |
| /usr/bin/node | 23556 | 23504 | 52 |
| /lib/x86\_64-linux-gnu/libc-2.28.so | 1384 | 1360 | 0 |
| /lib/x86\_64-linux-gnu/libpthread-2.28.so | 116 | 108 | 8 |
| /lib/x86\_64-linux-gnu/libgcc\_s.so.1 | 96 | 88 | 8 |
| /lib/x86\_64-linux-gnu/libm-2.28.so | 700 | 692 | 8 |
| /usr/lib/x86\_64-linux-gnu/libstdc++.so.6.0.25 | 1300 | 1252 | 48 |
| /lib/x86\_64-linux-gnu/librt-2.28.so | 32 | 24 | 8 |
| /lib/x86\_64-linux-gnu/libdl-2.28.so | 16 | 8 | 8 |
| /lib/x86\_64-linux-gnu/ld-2.28.so | 164 | 156 | 8 |
| **Totaal** | **27364** | **27192** | **148** |

Zoals verwacht is er dan ook een kleinere impact bij 2 containers maar shared memory is nog altijd significant groter dan private.

# Conclusie

Mijn metingen komen dus overeen met hun resultaten en valideert dus het onderzoek. Dit is dus een extra kostenefficiëntie maatregel en kan motiverend gelden voor ons onderzoek.

# Bibliografie

|  |  |
| --- | --- |
| [1] | J. B. Ferreira, M. Cello en J. O. Iglesias, „More Sharing, More Benefits? A study of Library Sharing,” in *Euro-Par 2017: parallel processing*, New York, NY, 2017. |
| [2] | „Linux-Containers,” [Online]. Available: https://linuxcontainers.org/lxc/getting-started/. |
| [3] | „PMAP,” [Online]. Available: https://linux.die.net/man/1/pmap. |
| [4] | „Proc manpage,” [Online]. Available: http://man7.org/linux/man-pages/man5/proc.5.html. |
| [5] | „OverlayFS,” Docker, [Online]. Available: https://docs.docker.com/storage/storagedriver/overlayfs-driver/#overlayfs-and-docker-performance. |
| [6] | „GSL usage,” GNU, [Online]. Available: https://www.gnu.org/software/gsl/doc/html/usage.html. |
| [7] | Sameo, „RSS, PSS, Clean vs Dirty, Private vs Shared,” 2017. [Online]. Available: https://gist.github.com/sameo/d49c50772d616ae00e96c9967e676976. |
| [8] | V. S, „RSS/PSS/USS,” Stackoverflow, [Online]. Available: https://stackoverflow.com/questions/22372960/is-this-explanation-about-vss-rss-pss-uss-accurate. |
| [9] | „Linux-AUFS,” [Online]. Available: https://www.thegeekstuff.com/2013/05/linux-aufs/. |