Create Simple Container

Ramil Khafizov

Advanced Total Virtualization

Innopolis University

2020

# Intro

The idea of the work is to understand the foundational thing in containerization technology - namespaces in linux. Getting familiar how clone() function can create child process with isolated network, filesystem etc, how cgroups can limit resources available to processes. After implementing all of this became much more clear.

## Links

This project on Github : https://github.com/smileyenot983/docker\_in\_c

## Discussing shortly what i did(also read comments in code)

1. Created new child process with new PID namespace, by calling function clone with NEWPID flag.

2. Created new namespace for child network by adding NEWNET flag and created virtual ethernet connection between child and parent processes and checked that they see each other using ping.

3. Created isolated filesystem namespace for child process, by creating zeroed file, putting it into a loop and mounting it as home for child filesystem. After that i created file in child filesystem and it was not visible from parent space.

4. I also isolated parent“s PIDS from child process by mounting proc folder inside child, thus creating new proc folder, which contains PIDS of all processes as files and wrote «ps aux» inside child namespace and verified that parent PIDS are not visible.

5. After that i created cgroups for memory and cpu for child pid and limited maximum amount of cpu and memory available for child process to be sure that it“s consumption will be bounded

6. I ran benchmark(sysbench) for container to test all main attributes – cpu, threads, memory, file input&output and it looks like container that i implemented has almost same performance as LXC and docker.

7. While doing this lab i learned more about containers, filesystem, system calls in linux

# Tests

To test my container against other products and host machine i have to measure cpu performance, memory performance, thread execution, reading and writing to a file.

## commands

|  |  |  |  |
| --- | --- | --- | --- |
| Metric | Sysbench command | Why this command | What is interesting in sysbench output |
| CPU total time [sec] | sysbench cpu --cpu-max-prime=20000 run | I increase cpu-max-prime because i expect to see more noticable difference between total time in different containers. Considered to be a good example, found from the following ​ source [[SysBenchExample]](https://www.howtoforge.com/how-to-benchmark-your-system-cpu-file-io-mysql-with-sysbench) | total time |
| Memory access | Sysbench memory run | To view physical memory performance | Almost 0 latency |
| FileIO read & write | Creating files: sysbench fileio --num-threads=16 --file-total-size=20G --file-test-mode=rndrw prepare  Testing read& write:  sysbench fileio --num-threads=16 --file-total-size=20G --file-test-mode=rndrw run  Deleting created files:  sysbench fileio --num-threads=16 --file-total-size=20G --file-test-mode=rndrw cleanup | I used total size higher than RAM available on my machine(8 gb) in order to be sure that this files are not in physical memory. | Interesting thing is that LXC has near native performance, while my container and docker“s performance near 2 times lower |
| Thread execution | sysbench threads --num-threads=64 --thread-yields=100 --thread-locks=2 run |  | High variance of results |

# Table With Metrics

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | host machine | my container | LXC (Ubuntu 20.04) | Docker(Ubuntu 20.04) |
| CPU [events/sec] | 478.20 | 475.16 | 468.34 | 471.51 |
| File IO Write  [writes/sec] | 3965.02 | 1810.15 | 3914.69 | 2069.63 |
| File IO Read  [reads/sec] | 5948.52 | 2715.97 | 5872.03 | 3104.94 |
| memory access [operations/sec] | 6174302.12 | 5895547.6 | 5073616.49 | 5031697.97 |
| Threads  [events/sec] | 6486.9 | 5861.9 | 5937.7 | 5047.7 |

I ran sysbench several times and results vary each time, so i just ran them one after another and wrote results on table.

# Explanation Why Metrics Differ

## CPU test - CPU performance is almost same for all cases

## File IO test – there is huge difference in FileIO between (host, LXC) and (docker,my container). One of the answers on stackoverflow says: «Layered filesystems are expensive». <https://stackoverflow.com/questions/21889053/what-is-the-runtime-performance-cost-of-a-docker-container>

Maybe that happens because docker manages image and container layers at the same time

<https://medium.com/@BeNitinAgarwal/docker-containers-filesystem-demystified-b6ed8112a04a>

## Memory test - there is small degradation of memory access for containers i think that“s because they are containers and while managing container, my machine still manages it“s own processes

## Threads test – only docker container has significantly lower threads execution performance and it has lower performance than LXC and my container almost at every run. I was not able to find any reason why that can happen.

# Sources

1. IBM reseach paper <https://dominoweb.draco.res.ibm.com/reports/rc25482.pdf> – in this paper they make comparison between host, docker and KVM. The general result: Docker is nearly identical to native performance.