POC - Proof of Concept Container Runtime implemented in Bash

Dmitrii Alekhin
Total Virtualization
Innopolis University
2024

Intro

The idea of the work is to implement a proof of concept container runtime with docker-like CLI written in Bash with networking support.

Features

- Supports several running containers
- Isolates PID, IPC, mnt (with procfs and sysfs), chrooted, and netns
- Docker-like simple CLI with ability to create, run, monitor and remove containers
- Dynamic ubuntu rootfs building including sysbench, python3, ping and iproute2 packages. Requires docker daemon running only for poc init command (for exporting rootfs.tar)
- Networking support through default network interface (parsed from route command) in isolated network namespace with veth-vpeer virtual interfaces. Implemented via iptables NAT forwarding
- Local both networking support through pocbr0 virtual bridge and with disabled networking

Links

This project on GitHub : https://github.com/dmfrpro/poc_container

Tests

To test my container against other products and the host machine I have to measure CPU, memory, and File I/O because they were stated in lab assignment and are primary measurable specs of a standard PC.

commands

Metric	Sysbench command	Why this command	What is interesting in sysbench output
CPU computation test	`sysbench cpu --threads=100 --time=60`	This command loads all CPU cores excessively (threads > logic cpus) with	total time

	`--cpu-max-prime=64000 run`	prime numbers computation which cannot be optimized from hardware side	
threads	`sysbench threads --threads=64 --thread-yields=100 --thread-locks=2 run`	It tests mutex performance which is meaningful in terms of testing performance in concurrent environment	total # of events, events avg and stddev
memory concurrent write test	`sysbench memory --threads=100 --time=60 --memory-oper=write run`	This command tests memory write access in a concurrent environment and also tests paging effectiveness	Memory speed
memory stress test	`sysbench memory --memory-block-size=1M --memory-total-size=10G run`	This command tests memory write access with continuously filling it up to 10G. This can be meaningful to spectate the system in OOM stage	Memory speed
fileio write test	`sysbench fileio --file-total-size=10G --file-test-mode=rndrw --time=120 --time=300 --max-requests=0 run`	This command performs large fileio test for 5 minutes in order to test fileio scheduling algorithm	Ops/sec (read, write, fsyncs), latency

Table With Metrics

Full metrics reports are available on project's GitHub:

1. Host: https://github.com/dmfrpro/poc_container/blob/main/report_host.md
2. POC: https://github.com/dmfrpro/poc_container/blob/main/report_poc.md

Explanation Why Metrics Differ

The fact that all metrics listed above differ from the benchmark measured on the host machine is connected with loop device isolation, which implies fileio syscalls to a loop device and then fileio syscalls to a hard drive (which contains an image file) second time. However, with default cgroups settings there are no CPU and memory limitations, thus corresponding tests show almost the same performance in comparison with host.

File IO test

POC has ~2-3 time slower ops/s (all read, write, fsync) and throughput. Avg latency is ~4 times larger than host, and Max latency is ~20 times larger than host.

Sources

1. [SysBenchExample] - “How to Benchmark Your System (CPU, File IO, MySQL) with Sysbench” https://www.howtoforge.com/how-to-benchmark-your-system-cpu-file-io-mysql-with-sysbench