Manninen, Harri; Jääskeläinen, Vesa; Blech, Jan Olaf

Performance Evaluation of Containerization Platforms for Control and Monitoring Devices

Published in:
Proceedings of the 25th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2020

DOI:
10.1109/ETFA46521.2020.9211901

Published: 01/09/2020

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Please cite the original version:
Manninen, H., Jääskeläinen, V., & Blech, J. O. (2020). Performance Evaluation of Containerization Platforms for Control and Monitoring Devices. In *Proceedings of the 25th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2020* (pp. 1061-1064). Article 9211901 (Proceedings IEEE International Conference on Emerging Technologies and Factory Automation). IEEE.
https://doi.org/10.1109/ETFA46521.2020.9211901

Performance Evaluation of Containerization Platforms for Control and Monitoring Devices

Harri Manninen
Aalto University
Espoo, Finland
harri.m.manninen@aalto.fi

Vesa Jääskeläinen
Vaisala Oyj
Vantaa, Finland
vesa.jaaskelainen@vaisala.com

Jan Olaf Blech
Aalto University
Espoo, Finland
jan.blech@aalto.fi

***Abstract*—Containerization platforms such as Docker are now common practice in the IT industry and are frequently used in combination with virtual machines in cloud-scenarios. Using containerization platforms for embedded control devices has so far been much less common. However, the advantages such as increased modularization and portability are of relevance for embedded control devices as well. On the other hand, embedded control devices are often characterized by a limited amount of computational resources and may face other constraints such as requirements on low power consumption. This paper presents first steps on estimating the resource needs for different containerization platforms. We present empirical work on resource consumption of containerization platforms for a Raspberry Pi-based control device and look at CPU usage, memory usage and power consumption.**

***Index Terms*—Containerization, embedded devices, performance**

I. Introduction

Application containers have become increasingly popular during the last decade in the IT sector. A containerized application typically includes the binaries of the software and a description of all its dependencies in a single, standalone package called a container image. Containers are used in virtualization scenarios (see e.g., [9]) where software is separated from the platform they are executed on. The term virtualization refers to the creation of a virtual version of a device or a resource such as a storage device or a network resource. Furthermore, the virtual part is typically isolated from the host system, which means that if it crashes, the host system is unaffected [8]. Containers can be easily deployed on many different systems, since containers cause no compatibility issues with required dependencies [1]. Using containers also has a few drawbacks: Application start-up times can be longer than running them natively. Containers can introduce some amount of overhead to the system, in terms of memory consumption and CPU usage.

In this paper, we are presenting some early quantitative results on the consumption overhead of container technologies in terms of CPU load, memory usage and power consumption for a Raspberry Pi-based embedded device.

II. Related Work

Several studies around the most used tools in containerization have been conducted. For example a performance analysis of Docker in specific environments is presented in [12] and [13], a Docker security analysis [17] and a Kubernetes performance analysis [14] is also available. Comparing performance between containers and virtual machines has been a common topic of research, see [9], [18], [19], [10], [11]. For example, [18] compares the performance between a Kernel-based Virtual Machine (KVM) and Docker containers in a server environment measuring factors such as memory and CPU usage as well as network and I/O bandwidth. It is concluded that Docker performance was equal or better in every test case. In [11] a study with similar results comparing the performance of a KVM and Docker in a Big Data environment was conducted. In [21] the performance of three containerization technologies is evaluated. The results show that all three container technologies have very similar and a near-native performance in terms of CPU, memory, disk and network usage. In [15] an investigation of the power consumption for Docker, LXC and two VM technologies is conducted.

In comparison to the work mentioned above, we are particularly focusing on embedded control devices, e.g., for use in an industrial environment. Few work has been done in this area: [20] investigates the feasibility of application containers in Docker in an embedded real-time Linux setting. Similar results are achieved in [16].

III. Platforms

This section describes the containerization platforms we have studied.

A. Docker

Docker's initial release was in 2013 [2]. It is generally regarded as a relatively mature technology supporting many platforms and being well documented. The most common use cases for Docker are cloud applications, however, Docker can also be used with desktops and embedded devices.

Docker uses a client-server architecture. The Docker Engine is built on a container runtime environment: containerd [3]. The Docker server communicates to the Docker client using a REST API. The Docker client provides a command line interface to the user. Docker features so called images which can be pulled from registries in a git-like fashion. Docker images can be built using éxtitDockerfiles. A *Dockerfile* contains

instructions that are needed for building a complete container image. Typically, a Linux distribution is used as a starting point when creating container images.

B. Balena

BalenaEngine is a lightweight container engine. It has been specifically built for embedded and IoT use cases. It has been designed for a small resource footprint, multi-architecture support and conservative memory usage. Balena is based on the Moby Project [5] created by Docker. This makes balenaEngine a lightwight version of Docker. The lightweightness is achieved by dropping Docker features that are not frequently used in the embedded world [4].

C. LXC

LXC is one of the oldest Linux container technologies. It has been under active development since 2008. This indicates a relative maturity. LXC consists of the *liblxc* library, a set of tools used to control containers, an API, and a set of templates used to create container images. LXC is rather lightweight and it can also be considered a rather low-level technology [6]. LXC's main focus is on system containers. However, LXC also supports the OCI image format, which allows for application containers.

D. Flatpak

Flatpak is a portable package manager for Linux. The focus is on sandboxing and isolating desktop applications. Flatpak can be used with a variety of CPU architectures, including ARM CPUs which are frequently used in embedded devices. While Flatpak has been strictly-speaking not developed as a containerization technology, it can serve a very similar purpose.

IV. Test Setup

This section describes our experimental setup: the environment and test applications.

A. Environment

Our experiments are carried out using the well-known embedded Linux device Raspberry Pi 4 (RPI). The Linux distribution that is used natively on the RPI is based on the Yocto Project [7], a tool used to build custom Linux distributions.

A shell script is used to log the CPU and memory usage. The script uses A program called *mpstat* to measure the CPU usage and *free* to measure the memory usage. *Mpstat* can be used to report the system's CPU usage for a defined period of time with a defined interval. In the script *mpstat* is invoked for a duration of three minutes with a measurement interval of one second. After that, only the average values are saved to a log file using *grep*.

Power consumption is measured using a LabJack T7 [22] data acquisition device. The measurement is done in the following way: (1) The voltage between two LabJack analog outputs with the Raspberry Pi installed between them is measured. (2a) The current measurement is done by first measuring the idle voltage between an analog input and ground with a shunt resistor between them. (2b), The Raspberry Pi is added to the circuit and the measurement is executed again. The difference between voltages in these two measurements is the current used by the Raspberry Pi. (3) The power consumption can then be calculated by multiplying voltage and current.

B. Test Programs

Three small, but IO intensive Python programs were written. The first program, uses serial communication to poll data from a separate Windows PC and writes the responses to a log file. The two other programs were used to communicate between separate containers with TCP. We believe that these programs capture essential characteristics of control and monitoring applications in an industrial embedded environment. Containers were created for each of the three programs. The Containers themselves are based on a minimal Linux distribution: Alpine.

V. Empirical Results

Figure 1 shows CPU usage, memory usage and power consumption for our application being deployed in a docker environment. Each figure shows the respective resource consumption for an idle system, a system running the above described applications natively (i.e., without container technology) and the system running the applications using docker. Note, that the docker server process (Unix daemon) is running in the background in all three cases, even if they are not used due to the application being executed natively or no application being executed at all (idle). The CPU usage shows the number and percentage of occupied cores, i.e., a value of two means that two cores have been fully occupied. Memory consumption is measured in KiB. The power consumption is measured in milliwats.

The experiments where conducted in several iterations after each other to get a feeling for possible initialization effects. This is represented through the x-axis. It can be seen that on average the resource consumption is only marginally higher when the application is executed in containers compared to a non-container execution of the application.

Figure 2 shows the respective results for Balena. Note that in all Balena cases a Balena background process is executed. For this reason the idle and native graphs show a different behavior compared to Docker.

Figure 3 show the results of the same experiments for LXC. The results are similar compared to docker.

Figure 4 shows the result for Flatpack. The memory consumption seems to indicate an underlying issue with Flatpack. This deserves further attention. Possible explanations comprise a memory leak in the underlying Flatpack processes (not in the application) or an issue with the initialization.

VI. Conclusion

We have presented first evaluation results for different containerization platforms for the use in embedded control and monitoring systems: we have studied Docker, Balena, LXC and Flatpak. In our evaluation we have focused on memory

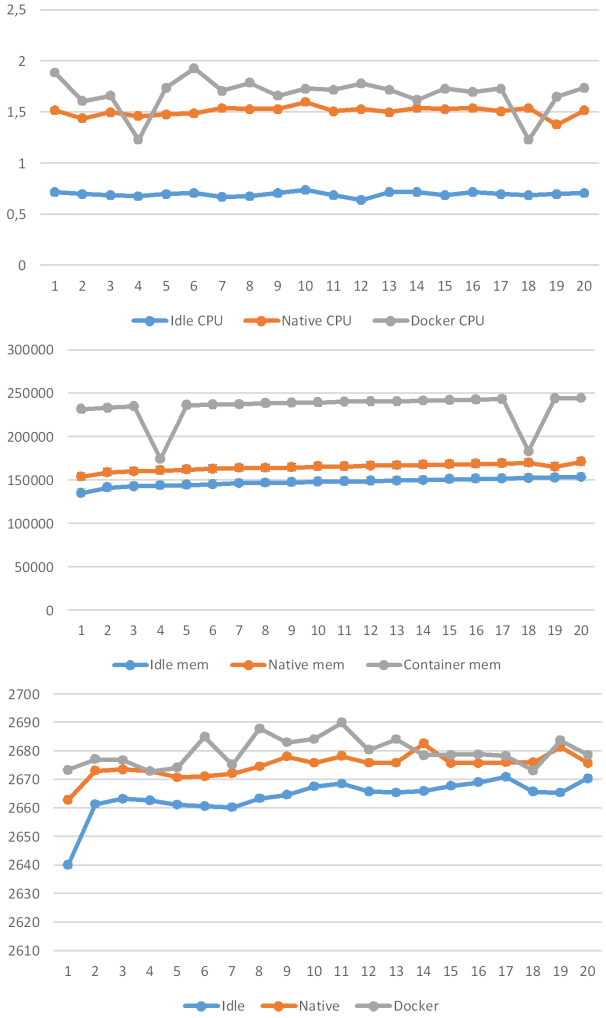

Fig. 1. CPU usage, Memory usage and Power consumption for docker

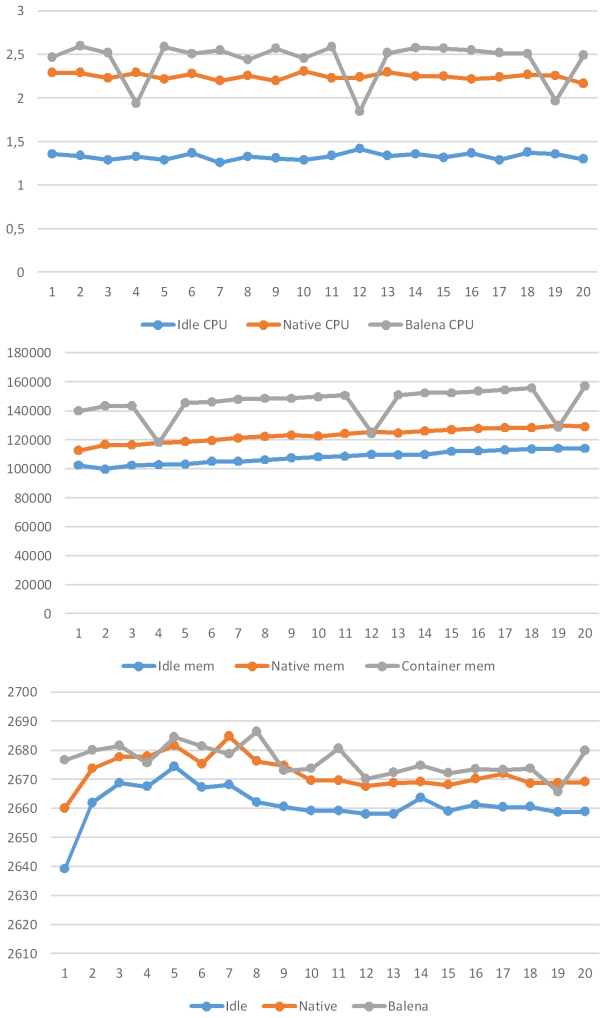

Fig. 2. CPU usage, Memory usage and Power consumption for Balena

usage, CPU usage and power consumption. It can be seen that on average containerization does not add much overhead in terms of CPU usage, memory and power consumption to an application. For this reason, we believe that containerization might be ready for embedded applications that run on platforms with characteristics that are similar to Raspberry Pi 4. Our applications are rather small and IO intensive, thus, we expect that there will be even less overhead for larger applications.

Future work will look at different applications and will also take different hardware platforms into account. In addition other containerization platforms may also be studied. The evaluation has shown certain unwanted behavior in some application scenarios. This will also deserve a further investigation.

References

[1] V. Noronha, M. Riegel, E. Lang, and T. Bauschert, "Performance Evaluation of Container based Virtualization on Embedded Microprocessors," in *2018 30th Int. Teletraffic Congr.*, Vienna, Austria, Sep. 2018, vol. 1, pp. 79-84, doi: 10.1109/ITC30.2018.00019.

[2] Docker blog. "5 years later, Docker has come a long way." docker.com. https://www.docker.com/blog/5-years-later-docker-come-long-way/ (accessed Mar. 24, 2020).

[3] Docker. "The Industry-Leading Container Runtime." docker.com. https://www.docker.com/products/container-runtime (accessed Mar. 24, 2020).

[4] A. Marinos. "Announcing balenaEngine: a container engine for IoT based on Moby technology from Docker". balena.io. https://www.balena.io/blog/announcing-balena-a-moby-based-container-engine-for-iot/ (accessed May 9, 2020).

[5] Mobyproject. "An open framework to assemble specialized container systems without reinventing the wheel." mobyproject.org http://mobyproject.org/ (accessed May 9, 2020).

[6] LXC. "What's LXC?" linuxcontainers.org. https://linuxcontainers.org/lxc/introduction/ (accessed Jan. 29, 2020).

[7] Yocto project. yoctoproject.org. https://www.yoctoproject.org/ (accessed Feb. 26, 2020).

[8] P. Sareen, "Cloud Computing: Types, Architecture, Applications, Concerns, Virtualization and Role of IT Governance in Cloud," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 3, no. 3, pp. 533-538, Mar. 2013.

[9] B. Bardhi, A. Claudi, L. Spalazzi, G. Taccari, and L. Taccari, "Virtualization on embedded boards as enabling technology for the Cloud of Things," in *Internet of Things,* R. Buyya and A. V. Dastjerdi, Eds., Cambridge, MA, USA: Morgan Kaufmann, 2016.

[10] A. M. Joy. "Performance comparison between Linux containers and virtual machines." *2015 Int. Conf. Adv. Comp. Eng. Appl.* Ghaziabad,

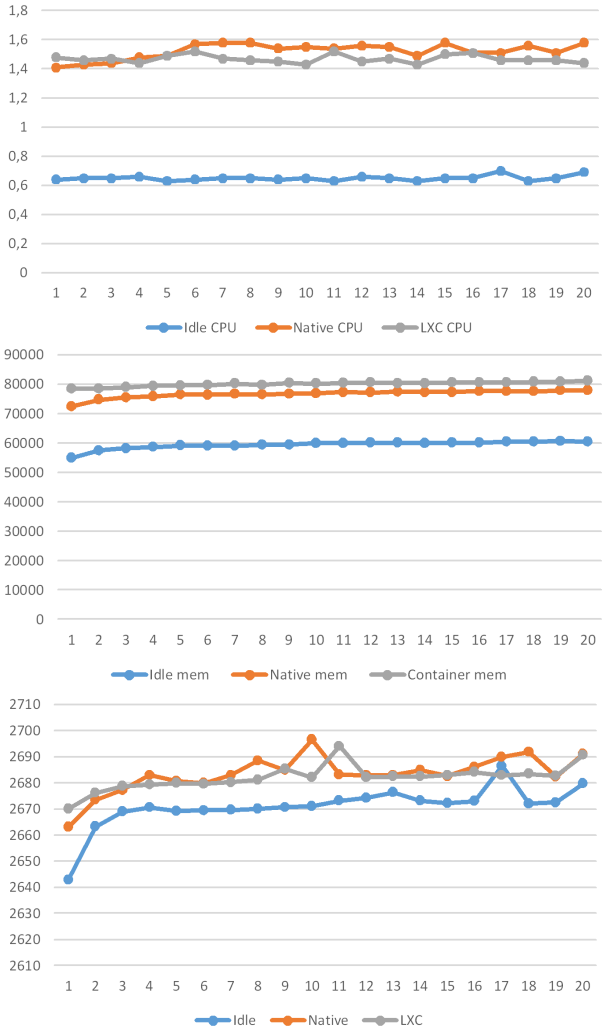

Fig. 3. CPU usage, Memory usage and Power consumption for LXC

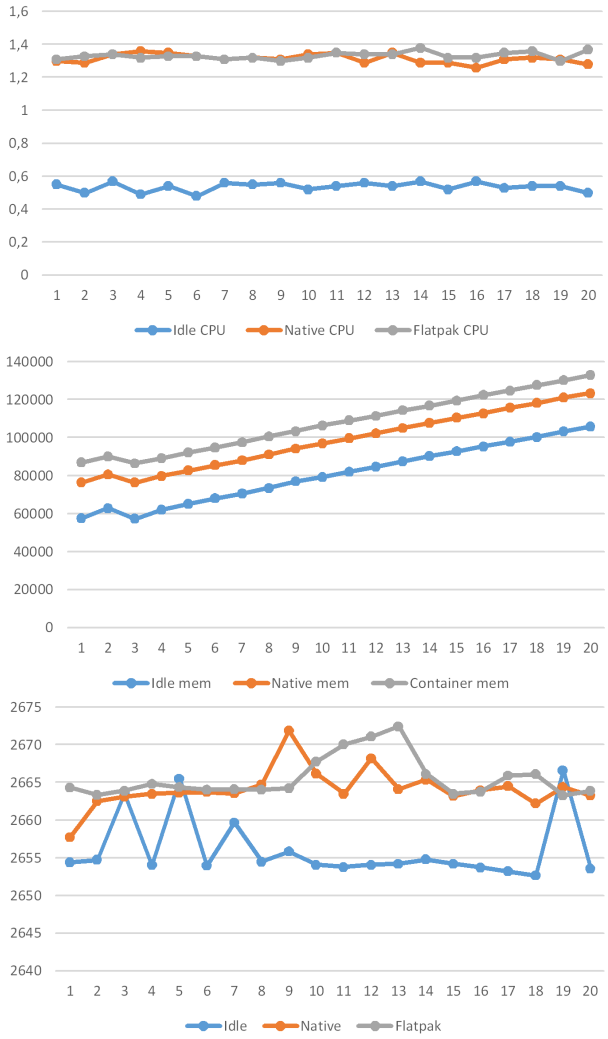

Fig. 4. CPU usage, Memory usage and Power consumption for Flatpak

India. Mar, 2015. pp. 342-346.

[11] Q. Zhang, L. Liu, C. Pu, Q. Dou, L. Wu, and W. Zhou. " A Comparative Study of Containers and Virtual Machines in Big Data Environment," in *2018 IEEE 11th Int. Conf. Cloud Comput.*, San Francisco, CA, USA, Jul. 2018, pp. 178-185, doi: 10.1109/CLOUD.2018.00030.

[12] P. D. Tommaso, E. Palumdo, M. Chatzou, P. Prieto, M. L. Heuer, and C. Notredame. The impact of Docker containers on the performance of genomic pipelines. *PeerJ*. Sep, 2015.

[13] M. T. Chung, N. Quang-Hung, M-T. Nguyen, and N. Thoai. "Using Docker in high performance computing applications." *2016 IEEE Sixth Int. Conf. Commun. Electron*. Ha Long, Vietnam. Jul, 2016. pp. 52-57. doi: 10.1109/CCE.2016.7562612.

[14] V. Medel, O. Rana, J. Á. Bañares, and U. Arronategui. "Modelling Performance & Resource Management in Kubernetes." *Proc. 9th. Int. Conf.* Utility Cloud Comp. Dec, 2916. pp. 257-262.

[15] R. Morabito. "Power Consumption of Virtualization Technologies: An Empirical Investigation," in *2015 IEEE/ACM 8th Int. Conf. Utility Cloud Comput.*, Limassol, Cyprus. Dec. 2016, pp. 522-527.

[16] R. Morabito. "A Performance Evaluation of Container Technologies on Internet of Things Devices," in *2016 IEEE Conf. Comput. Commun. Workshop*, San Francisco, CA, USA. Sep. 2016, pp. 999-1000.

[17] T. Bui, "Analysis of Docker Security," Aalto Univ., Espoo, Finland, 2014.

[18] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. "An Updated Performance Comparison of Virtual Machines and Linux Containers." IBM. Austin, Tx, USA. Res. Rep. RC25482 (AUS1407-001), Jul 2014.

[19] P. R. Desai. "A Survey of Performance Comparison between Virtual Machines and Containers," *Int. J. Comput. Sci. Eng*, vol. 4, no. 7, pp. 55-59. Jul, 2016.

[20] Lammi, T. "Feasibility of application containers in embedded real-time Linux," M.S thesis, Dept. Elect. Eng., Tampere Univ. Technol., Tampere, Finland, 2018.

[21] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange and C. A. F. De Rose. "Performance Evaluation of Container-based Virtualization for High Performance Comput. Environ.," in *2013 21st Euromicro Int. Conf. Parallel, Distrib., Netw.-Based Processing.*, Belfast, UK. Feb.-Mar. 2015, pp. 233-240.

[22] LabJack. "T7." labjack.com. https://labjack.com/products/t7 (accessed Mar. 14, 2020).