# Real-Time Containers: A Survey

**Václav Struhár**
Mälardalen University, Västerås, Sweden
vaclav.struhar@mdh.se

**Moris Behnam**
Mälardalen University, Västerås, Sweden
moris.behnam@mdh.se

**Mohammad Ashjaei**
Mälardalen University, Västerås, Sweden
mohammad.ashjaei@mdh.se

**Alessandro V. Papadopoulos**
Mälardalen University, Västerås, Sweden
alessandro.papadopoulos@mdh.se

## Abstract

Container-based virtualization has gained a significant importance in a deployment of software applications in cloud-based environments. The technology fully relies on operating system features and does not require a virtualization layer (hypervisor) that introduces a performance degradation. Container-based virtualization allows to co-locate multiple isolated containers on a single computation node as well as to decompose an application into multiple containers distributed among several hosts (e.g., in fog computing layer). Such a technology seems very promising in other domains as well, e.g., in industrial automation, automotive, and aviation industry where mixed criticality containerized applications from various vendors can be co-located on shared resources.

However, such industrial domains often require real-time behavior (i.e, a capability to meet predefined deadlines). These capabilities are not fully supported by the container-based virtualization yet. In this work, we provide a systematic literature survey study that summarizes the effort of the research community on bringing real-time properties in container-based virtualization. We categorize existing work into main research areas and identify possible immature points of the technology.

**2012 ACM Subject Classification** Computer systems organization → Real-time systems; Software and its engineering → Virtual machines

**Keywords and phrases** Real-Time, Containers, Docker, LXC, PREEMPT_RT, Xenomai, RTAI

**Digital Object Identifier** 10.4230/OASIcs.Fog-IoT.2020.7

**Funding** The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 764785, FORA – Fog Computing for Robotics and Industrial Automation.

## 1 Introduction

Fog Computing as well as cloud computing relies extensively on resource virtualization. In this area, the container-based virtualization is gaining its importance as a lightweight alternative of hypervisor-based virtualization. The container technology allows to execute applications and their software dependencies in a virtual environment independently on the software ecosystem of their hosts. A host can accommodate multiple containers at a time, providing means for container isolation and resource control for the containers. Container-based virtualization (sometimes referred as an OS level virtualization) does not require a hypervisor and therefore it provides near-native performance [13, 25], rapid deployment times and a low

overhead while still retaining a certain level of resource isolation and resource control. The containers are a de-facto standard for development of large scale web applications adopted by a number of companies [7].

The benefits of the container-based virtualization are aligned with the strive of the companies in other areas such as in industrial and robot control, automotive and aviation. In these industrial domains, there are strong requirements to (i) consolidate computational resources (Electronic Control Units, physical controllers) and (ii) provide a flexible environment for running (real-time) applications. Additionally, container-based virtualization can enable interruption-free hardware and software maintenance, dynamic system redundancy change and system redundancy healing [16]. However, in such fields stringent real-time requirements are often needed. This means that the applications inside of a container should meet predefined deadlines independently on other co-located containers.

In this survey, we summarize the research carried out in the area of real-time containers since the introduction of containers in Linux (i.e. 2008 [1]).

**The main contributions of this paper include:**

- Systematic literature survey of the real-time containers.
- Overview of the approaches and technology enabling real-time behavior of containers.
- Identification of pitfalls, challenges and future research directions for real-time containers.

## 2 The Review Process

The systematic literature survey is carried out with the guidance in [17]. The research questions are defined together with search queries and sources of the studies and, subsequently, we extract the data and answer to the questions. We apply the snowballing [28] method to identify relevant papers outside the search query. Databases used: *Scopus* and *IEEE*. Only full peer review papers published between 2008-2019 are considered. We search the databases using the following search queries:

(***Real-time*** OR ***RT***) AND (***Containers*** OR ***Container***)

The search string extracts 1855 articles in *Scopus* and 609 articles in *IEEE*. Out of that, we identify 38 and 23 potentially relevant articles by the title. As the number of articles is low, we fully screen each potentially relevant paper (abstract and full text) to make the decision for inclusion/exclusion into this survey. In total, we include 14 papers as seen in Table 1.

### 2.1 Question Formalization

In this work, we elaborate the following questions:

- **RQ1**: *Why and in which context have real-time containers been used?* Answering this question will give an overview of the motivation behind the use of real-time containers, expected benefits and areas where real-time containers are used.
- **RQ2**: *What approaches are used for enabling real-time behavior of containers?* The answer will give an overview of the approaches and technologies, their combinations and their usages for the real-time container-based virtualization.
- **RQ3**: *What are the pitfalls and weak points of using real-time containers that prevent full adoption of such technology in industry?* The answer for this question will give a list of research challenges and problems for real-time container computing.

# 3 Container-based Virtualization

From the runtime perspective, a container is a set of resource-limited processes that are isolated from the rest of the system and from other containers. This is achieved by utilizing two Linux kernel features: (i) Namespaces and (ii) Control groups (cgroups). Namespaces virtualize global resources (e.g., processes, network, inter-process communication) in the way that a group of processes can see and use one set of resources while another group can use different set of resources. Cgroups provide a mechanism for aggregating and partitioning sets of tasks, and all their future children, into hierarchical groups with specialized behaviour [2]. It allows to organize processes hierarchically and distribute system resources along the hierarchy.

## 3.1 Container Platforms

There are several container solutions, all of them rely on cgroups and namespaces. Thus, all the platforms pose similar options and performance [23]. The philosophy of using the two most commonly used container platforms LXC and Docker differs. Docker containers are microservice-based (each container should contain a single application), whereas LXC, similarly to Virtual Machines, allows to run a complex ecosystem of applications which is beneficial for emulation of legacy systems.

## 3.2 Real-Time Containers

The term real-time implies that the correctness of the system depends not only on the results of the computation but also on the time at which the results are produced [8]. Real-time systems can be categorized into three groups: hard, firm and soft real-time. Missing a deadline in a hard real-time system may cause catastrophic consequences, whereas missing deadline in a firm real-time system leads to the complete loss of the utility of the result. Missing deadline in a soft real-time system just degrades the utility of the result.

A real-time container is a container that provides resource isolation, resource control and additionally provides time deterministic and predictable behavior for the containerized application.

## 3.3 Real-time Support of Linux

To ensure the time predictable behavior of the containers, the operating systems must provide such capability. Default (Vanilla) Linux does not give any time guarantees on execution of tasks and therefore the predictability is low [24]. However, there are several approaches to improve the predictability: the real-time patch that improves preemptability of the Linux kernel and co-kernel approaches that run a real-time micro kernel in parallel to the Linux kernel. Containerized applications are scheduled in the same way as native applications using the host's scheduler, the Default Linux kernel provides three schedulers: *(i) Completely Fair Scheduler (CFS)*: Aims to maximize CPU utilization while also maximizing interactive performance. It does not give any time guarantees. *(ii) Real-Time scheduler (RT)*: The scheduler allows to schedule tasks in the fixed priority manner using First In First Out or Round Robin policies. The tasks run till they yield or are preempted by higher priority tasks. The Real-Time group scheduling [3] extension allows to divide and allocate CPU time between real-time and non real-time tasks. *(iii) Earliest Deadline First Scheduler (EDF)* Uses Constant Bandwidth Server [6] and allows to associate to each task a budget and a period.

# 4 Survey Results

In this section, we summarise relevant papers. There are four main directions for enabling real-time behavior of containers: (i) real-time patch based, (ii) co-kernel based, (iii) hierarchical scheduling based and (iv) custom approach. A short summary is provided in Table: 1.

## 4.1 Methods Based on PREEMPT_RT Patch

The real-time patch (PREEMPT_RT) improves the kernel's locking primitives to maximize preemptible sections. The advantage of the patch is that there is no need for special libraries or API needed by the application developers.

Moga et al. [22] considers real-time containers in the context of industrial automation systems that works with real-time data and have real-time deadlines on detection and response to events. The paper emphasises the need for OS-level virtualization in an industrial automation and gives examples of timing requirements of industrial applications (e.g. motor drive typically requires cycle time between 1ms to 250$\mu$s) and a need for synchronization between the containers. The evaluation of the effects of containers on performance of industrial automation systems is provided in two cases: (i) Cyclic behavior of a containerized application, (ii) Virtual networking performance for communications between containers. Cyclic behavior test evaluates the ability to execute application logic at pre-defined intervals, measures accuracy and jitter. Virtual networking test evaluates the ability to communicate between co-located containers in a time-bounded manner. The researches see the real-time container computing as a promising technology, however communication mechanisms between containers are not clear.

The work in [16] (and previously [15]) addresses a container based architecture for real-time controllers that allow a flexible function deployment and a support of legacy control applications. Such architecture is needed to preserve a functionality of legacy control programs and to reduce maintenance cost of legacy systems (in which the software is often bounded to a specific hardware and software ecosystem). The researchers investigate the feasibility of building a real-time capable system (for legacy systems) based on real-time containers, they target PLCs and automation controllers with the cycle time between 100ms to 1s. They perform a set of tests under various load scenarios (i) using containerized applications inside of Docker and (ii) running complete operating system (PowerPC) inside LXC. They conclude that a containerized execution of control applications can meet requirements of PLCs and automation controllers.

From the latency point of view, Masek et al. [21] performed a literature review on sandboxed real-time software on the example of self-driving vehicles. The researchers were interested in the question: How does the execution environment influence the scheduling precision and input/output performance of a given application? The result shows that docker does not impose additional overhead (similarly to [19]) for scheduling and input/output performance. However, selecting the correct kernel has a greater impact on the scheduling precision and input/output performance of containers.

Mao et al. [20] uses real-time containers to enable software-based RAN (Radio Access Network) in order to avoid high capital and operating expenditures during deployment of new standards. However, the software based RAN has strict deadlines to satisfy (1ms). The researchers use real-time patch to decrease the latency, interestingly they improve the latency 13.9 times by applying the patch in comparison to the vanilla Kernel.

**Table 1** Summary of studies elaborating on real-time containers.

| Study | Main focus | Approach & Technology | Communication aspects |
|---|---|---|---|
| Cinque et al. [9, 10] | ▬ Architecture definition<br>▬ Faulty tasks monitoring<br>▬ Implementation details | ▬ RTAI<br>▬ Docker<br>▬ Fixed priority scheduling | – |
| Cucinotta et al. [5, 11, 12] | ▬ Temporal Interference between containers | ▬ Hierarchical Scheduling | – |
| Tasci et al. [26] | ▬ Architecture definition<br>▬ Real-time communication between containers | ▬ Combination of Real-Time patch and Xenomai<br>▬ Docker | Design of messaging system based on ZeroMQ. |
| Moga et al. [22] | ▬ Feasability study<br>▬ Communication between containers<br>▬ Communication overheads | ▬ Docker<br>▬ Real-Time patch | Network performance and overhead measurements between containers using default Docker Linux NAT Bridge. |
| Hofer et al. [18] | ▬ Experimental comparison between Real-Time patch, Xenomai, Vanilla Linux | ▬ Real-Time patch<br>▬ Xenomai<br>▬ Vanilla Linux | – |
| Goldschmidt et al. [15, 16] | ▬ Architecture definition<br>▬ Feasibility study | ▬ Real-Time patch<br>▬ Legacy systems emulation in real-time containers | – |
| Telschig et al. [27] | ▬ Model-based architecture and analysis<br>▬ Dependable real-time container computing. | ▬ LXC | – |
| Mao et al. [20] | ▬ Minimizing latencies in software-based Radio Access Networks | ▬ Docker<br>▬ Real-Time patch | Application of fast packet processing using Intel Data Plane Development Kit. |
| Masek et al. [21] | ▬ Systematic evaluation of sandboxed software | ▬ Real-Time patch | – |
| Wu et al. [29] | ▬ Dynamic CPU allocation for mixed-criticality real-time systems | ▬ Custom scheduling mechanism<br>▬ Docker | – |

## 4.2 Methods based on Real-time Co-Kernel

In this approach, a real-time micro-kernel runs in parallel to Linux kernel. The real-time co-kernel handles time critical activities (e.g., handling interrupts and scheduling real-time threads), standard Linux kernel runs only when the co-kernel is idle. In comparison to the real-time patch, the co-kernel approach offers lower latencies and lower jitter. On the other hand, it requires special APIs, tools and libraries for the application development. Additionally, there are impediments with scaling co-kernel solutions on large platforms (e.g., many cores platforms). There are two co-kernel alternatives: Real Time Application Interface (RTAI) and Xenomai.

RTAI aims to minimize latencies to the lowest technically possible values. Real-time tasks are compiled as kernel modules and ran in the kernel space. Xenomai [14] is a fork of RTAI. Its mission is to enable real-time tasks in the user space. It consists of an emulation layer that is capable of reusing code from other RTOSes.

Tasci et al. [26] elaborates on modularization of real-time control applications into real-time containers. Such modular architecture needs two essential parts: (i) Computational part, enabled by a real-time operating system (combination of Xenomai and real-time patch), and (ii) Messaging part that allows passing messages between containers in a real-time manner. Traditional monolithic architectures communicate through function calls and shared memory, the containers do not make the assumption if they are running on the same host or in a distributed environment (they communicate through standard OS networking stack), therefore direct passing messages through shared memory is not directly supported. Hence, the researchers provide a design and implementation of a custom made real-time messaging system for containers based on ZeroMQ [4].

Hofer et al. [18] use the real-time containers in the context of control applications. The paper presents comparison between type 1 hypervisor, Vanilla Linux, Xenomai co-kernel and Linux with real-time patch for various idle and stress scenarios.

## 4.3 Method Based on Hierarchical Scheduling Of Containers

Inspired by a similar concept in the hypervisor-based virtualization where a global scheduler assigns CPU time for the virtual machines, the second layer scheduler schedules the individual tasks of the VM.

Cucinotta, Abeni et al. [5, 11, 12] proposed the use of real-time containers on the field of Network Function Virtualization (NFV), where the functionality of traditional physical network devices (e.g., firewalls) is transformed into software components (in containers) that are consolidated in a single computing device. NFV has critical latency requirements inducted by the need of time critical per-packet processing. The researchers modified the Linux scheduling mechanism to provide two levels hierarchical scheduling. First level Earliest Deadline First scheduler selects the container to be scheduled on each CPU. Subsequently the second level Fixed Priority scheduler selects a task in the container. CPU reservation (runtime quota and period) is assigned to each of the containers.

## 4.4 Custom Methods

Wu et al. [29] proposed the Flexible Deferrable Scheduler for containerized mixed-criticality real-time systems that consist of real-time and non real-time containers. The scheduler guarantees the allocated CPU capacity to real-time containers and dynamically distributes the unused capacity to non real-time containers. The work supplements Completely Fair Scheduler with a Workload Adjustment Module that collects CPU utilization by containers and Dynamic Adjustment Module that allocates CPU to the container.

Cinque et al. [10] (previously [9]) implemented real-time containers using Linux patched with real-time co-kernel (RTAI) and utilizing custom made monitoring and policy enforcing modules. Their solution allows to co-habit containers with different criticality levels and to prevent fixed-priority hard real-time periodic tasks inside of the containers to affect the temporal guarantees of other containers. The temporal guarantees are provided by two mechanisms: (i) proper tasks priority assignments to tasks inside the containers and (ii) monitoring and enforcing temporal protection policies. The former ensures that tasks inside of the high-criticality containers are assigned higher priorities than tasks in the lower-criticality containers and thus they are never preempted by tasks of lower criticality containers. The latter monitors the tasks and, in case of overruns or overtimes, it enforces one of the temporal protection policy (i.e., kill or suspend the faulty task, suspend the task until the next period).

## 5 Challenges of Real-time Container-based Virtualization

In the reviewed papers, we identified shortcomings and immature aspects of real-time container virtualization that prevents the expansion of the technology. Below, we listed them categorized in three groups: (i) tools support, (ii) real-time communication support, and (iii) miscellaneous.

**Lack of tools for real-time container management.** The reviewed papers emphasises a need for supporting tools for real-time containers. Tools that enable deployment on containers taking into account real-time requirements of containers and properties of computational nodes.

- The need for an orchestration tool that can schedule real-time containers based on pre-configured capabilities [18].
- Middleware that is aware of both communication needs as well as run-time and performance isolation needs [22].
- Framework to expose the runtime requirements of real-time application running inside containers and to enforce an optimal allocation of containers to resources [22].

**Communication between real-time containers.** Real-time communication between a container and its environment has to be further researched. Currently, the reviewed papers emphasize the following issues:

- Need for a real-time communication among containers [22].
- Further investigation on container security restricted container access and intra-container communication [18].
- A research on data management shared across containers [15].

**Miscellaneous.** In addition to generic issues that may harm the real-time behaviour (e.g., shared caches, memory and I/O), the studies reviewed highlight the following points and questions:

- Lack of safety, security analysis of real-time containers and vulnerability management for the acceptance in industry [15, 27].
- Lack of latency and performance tests of recent releases of a patched Linux Kernel. As well as a proper analysis of configuration of the Linux kernel parameters that may improve overall task determinism. [18].

- The measurements of memory overhead of the container solution and is it acceptable for real world applications [15].
- Processes in different containers may use the same resources in the same way because of their independent views of the system (i.e., processes are not aware of a resource-limited isolated environment co-located with other containers). This results in poor resource utilization as well as a potential violation of the real-time execution assumptions [22].
- Container approaches are a new technology. Will this create problems due to its possible immaturity [15]?

## 6 Conclusion

Container-based virtualization has become popular as a lightweight alternative of hypervisor-based virtualization. The technology has proven its viability in large cloud-based systems, it has been adopted by a number of enterprise companies and it is supported by a large scale of tools (e.g., container orchestration and monitoring tools).

However, in industrial domains where the real-time behavior is required, the container-based virtualization seems not to be mature enough. In this paper, we summarize the research carried out in the field of real-time containers. We show in what contexts, what approaches and technologies are used, and what are the possible immaturity points of the real-time container-based virtualization.

## References


1 Notes from a container. URL: `https://lwn.net/Articles/256389/`.

2 The Linux Kernel Archives. URL: `https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt`.

3 The Linux Kernel Archives. URL: `https://www.kernel.org/doc/Documentation/scheduler/sched-rt-group.txt`.

4 Zero MQ. URL: `https://zeromq.org/`.

5 Luca Abeni, Alessio Balsini, and Tommaso Cucinotta. Container-based real-time scheduling in the linux kernel. *SIGBED Rev.*, 2019.

6 Luca Abeni, Giuseppe Lipari, and Juri Lelli. Constant bandwidth server revisited. *Acm Sigbed Review*, 2015.

7 Thanh Bui. Analysis of docker security. *ArXiv*, abs/1501.02967, 2015. `arXiv:1501.02967`.

8 Giorgio C Buttazzo. *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer Science & Business Media, 2011.

9 Marcello Cinque and Domenico Cotroneo. Towards lightweight temporal and fault isolation in mixed-criticality systems with real-time containers. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2018.

10 Marcello Cinque, Raffaele Della Corte, Antonio Eliso, and Antonio Pecchia. Rt-cases: Container-based virtualization for temporally separated mixed-criticality task sets. In *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*, 2019.

11 Tommaso Cucinotta, Luca Abeni, Mauro Marinoni, Alessio Balsini, and Carlo Vitucci. Virtual network functions as real-time containers in private clouds. In *IEEE CLOUD*, 2018.

12 Tommaso Cucinotta, Luca Abeni, Mauro Marinoni, Alessio Balsini, and Carlo Vitucci. Reducing temporal interference in private clouds through real-time containers. In *2019 IEEE International Conference on Edge Computing (EDGE)*, pages 124–131, 2019. `doi:10.1109/EDGE.2019.00036`.

**13** W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. An updated performance comparison of virtual machines and linux containers. In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2015.

**14** Philippe Gerum. Xenomai-implementing a rtos emulation framework on gnu/linux. *White Paper, Xenomai*, pages 1–12, 2004.

**15** Thomas Goldschmidt and Stefan Hauck-Stattelmann. Software containers for industrial control. In *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2016.

**16** Thomas Goldschmidt, Stefan Hauck-Stattelmann, Somayeh Malakuti, and Sten Grüner. Container-based architecture for flexible industrial control applications. *Journal of Systems Architecture*, 84:28 – 36, 2018. URL: `http://www.sciencedirect.com/science/article/pii/S1383762117304988`, `doi:https://doi.org/10.1016/j.sysarc.2018.03.002`.

**17** Jo Hannay, Dag Sjøberg, and Tore Dybå. A systematic review of theory use in software engineering experiments. *Software Engineering, IEEE Transactions on*, 2007.

**18** Florian Hofer, Martin Sehr, Antonio Iannopollo, Ines Ugalde, Alberto Sangiovanni-Vincentelli, and Barbara Russo. Industrial control via application containers: Migrating from bare-metal to iaas. In *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 62–69, 2019. `doi:10.1109/CloudCom.2019.00021`.

**19** A. Krylovskiy. Internet of things gateways meet linux containers: Performance evaluation and discussion. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, 2015.

**20** C. Mao, M. Huang, S. Padhy, S. Wang, W. Chung, Y. Chung, and C. Hsu. Minimizing latency of real-time container cloud for software radio access networks. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, 2015.

**21** Philip Masek, Magnus Thulin, Hugo Andrade, Christian Berger, and Ola Benderius. Systematic evaluation of sandboxed software deployment for real-time software on the example of a self-driving heavy vehicle. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, 2016.

**22** Alexandru Moga, Thanikesavan Sivanthi, and Carsten Franke. Os-level virtualization for industrial automation systems: are we there yet? In *SAC '16*, 2016.

**23** Roberto Morabito, Jimmy Kjällman, and Miika Komu. Hypervisors vs. lightweight virtualization: a performance comparison. In *2015 IEEE International Conference on Cloud Engineering*, 2015.

**24** Claudio Scordino and Giuseppe Lipari. Linux and real-time: Current approaches and future opportunities. In *IEEE Internafional Congress ANIPLA*, 2006.

**25** Cristian Spoiala, Alin Calinciuc, Cornel Turcu, and Constantin Filote. Performance comparison of a webrtc server on docker versus virtual machine. *13th International Conference on DEVELOPMENT AND APPLICATION SYSTEMS, Suceava, Romania, May 19-21, 2016*, 2016.

**26** Timur Tasci, Jan Melcher, and Alexander Verl. A container-based architecture for real-time control applications. In *2018 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*. IEEE, 2018.

**27** Kilian Telschig, Andreas Schonberger, and Alexander Knapp. A real-time container architecture for dependable distributed embedded applications. *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 1367–1374, 2018.

**28** Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, New York, NY, USA, 2014. Association for Computing Machinery. `doi:10.1145/2601248.2601268`.

**29** J. Wu and T. Yang. Dynamic cpu allocation for docker containerized mixed-criticality real-time systems. In *2018 IEEE International Conference on Applied System Invention (ICASI)*, 2018.