On container usability in large-scale edge distributed system

Miloš Simić, Milan Stojkov, Goran Sladić and Branko Milosavljević, Miroslav Zarić
Faculty of Technical Sciences, Novi Sad, Serbia
milos.simic@uns.ac.rs

***Abstract*—Containers are not new technology. They emerge in the late seventies, but they become more popular in the past decade in various systems and companies to achieve high resource utilization, efficient task-packing, process-level isolation, etc. Those container properties allow large internet companies like Google, Amazon, and Facebook to satisfy an ever-growing number of users who use their applications, services, and big data workloads. These companies can run dozens of tasks at scale, to satisfy all their operational needs. On the other hand, developers can create more efficient software and run it everywhere with the same behavior, almost without any additional work. With emerge of automatic cluster management and cluster schedulers, these containers get used even more on a day to day jobs, making them as easy as possible. In this paper, we present a review of usage of container technologies in the edge computing systems, in domains with constant data flow with processing on the very edge of the network.**

Keywords: distributed systems, big data, service-oriented architectures, cloud computing, edge computing, orchestration, containers

I. INTRODUCTION

Nowadays, we are facing a massive movement in processing away from the standard, centralized computing model that is provided through cloud computing paradigm, especially in the domain of Internet of Things (IoT) where we have a lot of sensors that are constantly sensing some events.

This movement returns the distribution of computing power and logic to the edge of the network. Edge network is the idea of connecting sensors to programmable automation controllers (PAC) which can then handle processing, storage, and communication. The basic concept of edge computing is to leverage new generation technologies, processes, services, and applications that are built to take advantage of this new infrastructure. The key difference with this model is that it operates and it is deployed on computing hardware closer to the edge of the network. Thus, it is bringing the cloud computing model closer to the ground.

With the architecture of distributed edge devices [1], we also need to consider how to manage them and orchestrate jobs on those devices for best resource utilization. Since containers are now almost a standard in large-scale distributed systems and in the cloud, we can investigate if they are applicable in the edge computing architecture, with resource restricted hardware like ARM devices (ex. Raspberry Pi, Beaglebone, etc.).

Containers provide us a packaging mechanism in which applications can be abstracted from the environment in which they run. This decoupling allows container-based applications to be deployed easily and consistently, in different environments like a data center, the public cloud, or even a user personal computer. Containerization provides a separation of concerns [2], as developers focus on their application logic and dependencies, while operation teams focus on deployment and management without knowing application details such as specific software versions and configurations specific to the application.

The concept of containers was introduced in 1979 with UNIX *chroot* command. This is an operating system command for changing the root directory of a process and its children to a new location in the file-system which is only visible to a given process. In 2000, Derrick Woolworth introduced *FreeBSD Jail*. It is an operating system call, similar to *chroot*, but it includes additional features for isolating the filesystem, users, networking, etc. That means assigning an IP address for each jail. 2005 *OpenVZ* makes use of a Linux kernel for providing virtualization, isolation, resource management, and checkpointing. Each *OpenVZ* container would have an isolated file system, users and user groups, a process tree, network, devices.

Real breakthrough happened when Google added *control groups* mechanism or *cgroups* for short. *Cgroups* was implemented at Google in 2006 and added to the Linux Kernel in 2007, for limiting, accounting, and isolating resource usage such as CPU, memory, disk I/O, network, etc. for a processes collection. This shows how early Google was involved in container technology, and how containers help them to build more efficient and robust systems. Linux Containers (*LXC*) [3][4] is the first, most complete implementation of Linux container manager and it is introduced in 2008. It was implemented using *cgroups* and Linux *namespaces*. *LXC* provided language bindings for Python, Lua, Go, Ruby, and Haskell. *LXC* works on base Linux kernel without requiring any patches.

Despite all these great efforts and technologies, containers did not get in massive usage until the release of modern container frameworks such as Docker and Rocket. Before those technologies, usage of LXC containers was not that easy. It required a great amount of knowledge and users tend not to use them that often.

When *Docker* was released it provided a more user-friendly interface for users to interact with containers. Rocket, developed by *CoreOS*, tend to fix some of the drawbacks found in *Docker*. In 2016, Microsoft took an initiative to add container support to the *Microsoft Windows Server* operating system, called *Windows Containers* [5]. In the further text, we use word containers to represent *Docker* containers.

This paper is organized as follows. Section II presents edge computing challenges authors will analyze from containers point of view. Section III presents the container design and security aspects of containers. Section IV makes a comparison between containers and virtual machines. Section V summarizes conclusions and briefly propose ideas for future work.

II. EDGE COMPUTING CHALLENGES

Since edge computing shifts the architecture towards more decentralized one, different challenges arise. The business logic and data that is being processed are pushed away from the cloud eliminating cloud services as the bottlenecks and potential single points of failure. Even though in this way we can reduce response time and improve Quality of Service, we identify some challenges that must be tackled:

- Service deployment – edge devices are often inferior when it comes to performance or capacity compared to servers in the cloud. This constraint should be taken into consideration and try not to affect the way to easily install, manage, upgrade and terminate running services [11]
- Service management – this includes service registry, discovery, and orchestration. If we presume that devices in edge computing can constantly join or leave the network, there must be a way to find the compatible device on which the service can be deployed and that can run that service with minimal or no degradation in performance at all.
- Service robustness and recovery plan – the platform on which services are running must be fault tolerant and there must be defined lightweight recovery plan. A study by D. LeClair suggested that edge computing availability should surpass the typical three nines (99.9%) of the data center. It suggests edge computing should support at least five (99.999%) availability on the edge [11].
- Data caching – the response time can be improved by caching data on the edges. Since container images are made of layers of images, these layers are reusable. This is a great ability that means that we can save disk space, and be able to faster start services on nodes.
- Service monitoring - these devices are more prone to failure than regular machines in the cloud. We must have a proper way to monitor nodes and services running on those nodes. At all time we must have insight what is happening in our edge cluster in order to react fast, reschedule jobs on other nodes and heal the cluster or to find performance bottlenecks, which nodes are slow, and again do rescheduling of the jobs on other nodes. This ability is strongly connected to service orchestration since those two things should collaborate in order to fully automate the rescheduling process.
- Service configuration must be fully automated since we are talking about a big number of nodes in edge clusters and even more services. We must have a strategy to change configuration or secrets and keys at any time in an efficient way - either by parameterization with configuration files or environment variables and change the overall behavior of services.

Containers can offer a solution for these challenges since they are very similar to how it is done in the cloud computing systems. Authors in [12] show positive impact running applications in the containers on the ARM devices, multiplatform Docker images [13] and running orchestration tools like Kubernetes [14][17], enforce us to propose that multiple ARM devices can be connected and form micro datacenter [1]. This data center will mimic real datacenter but running specific user-defined applications inside containers. These applications are specifically designed for edge computing, and they will do filter and preprocess of the data before sending it to the cloud. These edge applications can come in the form of: 1) streams, that should continually do some processing on data as it is arriving(long-running jobs), 2) standard batch jobs that do some batch processing over some collection of data and 3) reacting only on if some value passes some threshold in the form of events. This model could be used to help cloud computing by filtering and preprocessing data at the edge on the network, and only send valuable data to the future analysis. This could accelerate time to market for the users and also save money on storing and processing unnecessary data.

In the rest of the paper, we will discuss how do containers differ from heavyweight virtual machines in terms of solving these issues.

Containers show more benefits than traditional VMs. But despite those benefits, there are some problems that must be addressed. Problem with the approach of using only containers for the infrastructure and the applications is that we inherit all current and future problems that could possibly come with the containers. Not like real data centers, where we can choose between VM, containers or some hybrid solution, and use best for the specific use case.

III. CONTAINER DESIGN

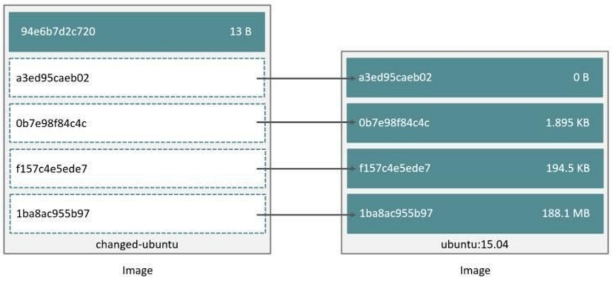
1. *Technical aspects*

As we previously mentioned, containers are based on a few technologies that allow various kind of isolations. (Docker) containers run on top of a few important technologies that allow users to run their containers isolated from the rest of the application virtually anywhere and in a predictable manner. Technologies that allow this behavior are:

Namespaces. It is a concept originally developed by *IBM*. A namespace, in Linux terminology, wraps a set of system resources and presents them to processes within that namespace, making it look as if they are dedicated to the processes. This means that any process cannot see resources and interact with other processes and their resources (mounted drives, child processes, network, etc.).

Cgroups. It is a Linux kernel concept that allows the isolation and usage of system resources, such as CPU, memory, network, disk I/O, for a group of processes. For example, if we have an application that is taking too many resources, we can put that application into cgroup to limit its resource usage. A user specifies how much resource each application is able to get. *Cgroups* use a pseudo-filesystem to expose metrics for each container, making them available to collect, which simplify monitoring and metrics collection.

UnionFS (union file systems) [8][9]. It operates by creating layers, making them very lightweight and fast. Containers bring immutable (unchanging over time) software in the form of layers. During build time, multiple immutable layers of software are stacked to get the desired applications with their dependencies. This process creates a container image that acts as a template for creating a container. Using a union filesystem allows each layer that is created to be reused by an unlimited number of images. This saves a lot of disk space and allows images to be built faster since it is just re-usage of an existing layer. Additionally, the read/write top layer gives the appearance that you can modify the image, but the read-only layers below actually maintain their integrity of the container by isolating the contents of the filesystem. Picture 3 show re-usage of existing images on disk.



Picture 3. Re-usage of existing images on disk [https://bit.ly/2KiFaaO]

With these technologies, containers give us the ability to start and stop applications (processes) fast, not starve system for resources since there is resource isolation and give us the ability to share layers when creating applications which means even faster start and smaller disk usage. On the other hand, since containers operate on kernel level and they virtualize kernel of the operating system we cannot run different operating systems in containers. Since containers are immutable we must be aware of how to persist data after a container shuts down. But building applications using containers give us an easier way to automatic container orchestration. Since containers are running in production for many years now people from Google developed a few patterns on how to run applications in containers [10].

2. *Security aspects*

Security implementation of containers is still unexplored field, but some features for making containers more secure already exist. The first recommendation for the edge environment is to create containers with the least privilege possible. Also, besides the aforementioned features such as namespaces and Cgroups, containers can utilize a few extra features:

Seccomp – secure computing mode profile can be associated with a container to restrict available system calls.

Security Enhanced Linux – it provides an additional layer of security to keep containers isolated from each other and from the host.

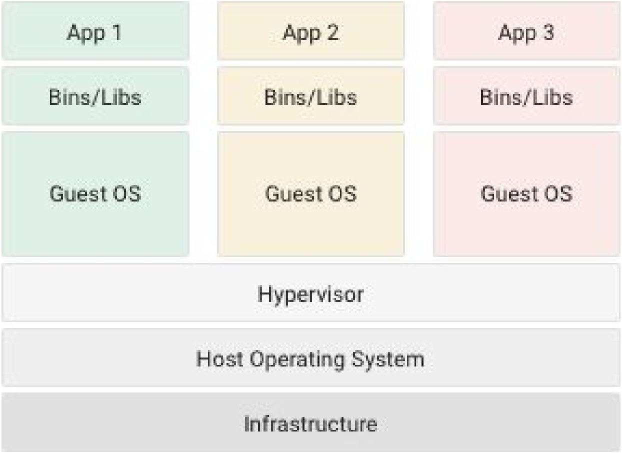
Linux capabilities – it can be used to lock down root in a container. Capabilities are distinct units of privilege that can be independently enabled or disabled. So, when running containers, users can drop multiple capabilities without impacting the majority of containerized applications.

Docker containers, as the most popular container technology, offer a new level of defense by separating the applications from the host. They have process restrictions, device and file restrictions, application image security, etc. Containerized versions of open source packages can be malicious and that is why Docker allows container images to originate from a certified, trusted, remote registries. Since multiple containers are used to deliver one application, container orchestration is a necessity. This necessity opens new questions regarding a safe communication that includes defining strong role-based access control for all the elements included in this communication. RedHat OpenShift offers some solutions for authentication and authorization as critical features for container platform.

Modern Linux kernels have many more security constructs in addition to the previously mentioned concepts of seccomp, SELinux, capabilities, namespaces, and Cgroups. Docker can leverage existing systems like TOMOYO, AppArmor, SELinux, and GRSEC, some of which come with security model templates available out of the box for Docker containers [15].

IV. CONTAINERS AND VIRTUAL MACHINES

Containers are usually compared to virtual machines (VMs). Virtual machines start a full guest operating system such as Linux or Windows for every VM instance we use. This guest operating system runs on top of a host operating system with virtualized access to the underlying hardware. The benefit of this approach is that we can use different operating systems at the same time since every virtual machine runs the full guest operating system independently. Picture 1 shows the virtual machine architectural diagram.

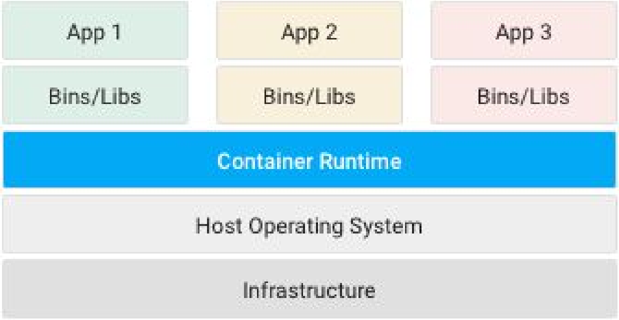


Picture 1. The architectural diagram of virtual machines [https://cloud.google.com/containers/]

Like virtual machines, containers allow users to package applications together with libraries and other dependencies, providing isolated environments for running software services. Similarities between those two technologies end there since containers offer a far more lightweight unit for developers and operational teams [4]. Instead of virtualizing the hardware, as we do with the virtual machines, containers provide virtualization at the operating system level.

Because they share the operating system kernel instead of running full guest operating system, containers start much faster [8] and use fewer resources compared to booting an entire operating system. With this ability, containers are bound to the specific operating system, unlike virtual machines.

This property of containers gives us an easier and faster way to deploy containerized applications to almost any type of hardware, unlike virtual machines. But also, all service dependencies are included in the container image. Since containers are lightweight, recovery from a crash or some other problem is much faster than virtual machines which boot the entire operating system, and then all the services. Every container is made of many different layers. These layers are shared between other containers and they are reusable, which mean that we already have data caching provided by default. We must always know in which state are our services that run inside containers, and how many resources they are using. *Cgroups* tracks system resources (like CPU, memory, and block IO) for all containers, and expose these metrics, making them available to collect, which make monitoring a bit easier. Picture 2 show containers architectural diagram.



Picture 2. The architectural diagram of containers [https://cloud.google.com/containers/]

There are several open source products that are dealing with container orchestration like Kubernetes [16], Docker Swarm, Hashicorp Nomad. These products are to a greater or lesser extent based on the Google Borg [17] system that runs the whole Google system. The idea behind these tools is that user submits configuration or deployment file in which he describes in detail how these orchestration engines need to run these services, and how much of them. Then, orchestration engine needs to find a node(s) (if available) and schedule those services on those nodes, monitor them, and take care that a specified number of services are always running. The user is in control of these services, which means that he can scale them up or down if needed. Some sort of container orchestration needs to be used in edge cluster also, in order to provide fully automated deployment of containers without user's interference.

V. CONCLUSION

Since we are facing massive shift from centralized computing architectures, we should find a new way to address problems with new applications like the internet of things area where we can collect and preprocess data at the very end of the network.

Containers come with great ability to pack and isolate applications that simplify the job for developers and operational teams. With these great options, containers can be applied to many different tasks. We saw how containers can help us in new challenges that edge computing brings. Things like service deployment, service management, service robustness and recovery, service monitoring, service configuration, and data caching can be achieved easier using containers. Unlike traditional virtual machines, containers are more lightweight, use fewer resources and services that are packed in containers are faster to start and stop. restart or even reschedule in different nodes using service orchestration.

With all these properties to run virtually everywhere almost without changes, process isolation, fast start and stop, and re-usage of existing layers of other containers we found that running applications on containers can benefit edge computing especially in the large-scale distributed environment in cooperation with some orchestration engine. Containers give us a lot of benefits compared to traditional VMs. Problem with the approach to use containers only, is that we inherit all the problems that could possibly come with the containers. Not like real data centers, where we can choose between VM, containers or some hybrid solution, and use best for the specific use case.

Future work should include research and development tools for monitoring of the edge computing system, but also service and system configuration elements to provide better automation in the large-scale edge computing systems. Also, the research should include the investigation of the possibility of unikernels [18][19] usage for edge computing applications with container tools [20].

REFERENCES

1. Simić, M., Stojkov, M., Sladić, G., Milosavljević, B. Edge computing system for large-scale distributed sensing systems. In: Konjović, Z., Zdravković, M., Trajanović, M. (Eds.) ICIST 2018 Proceedings Vol.1, pp.36-39, 2018
2. J. Bottomley, What is All the Container Hype?, Parallels and Linux Foundation, April 2014
3. D. Bernstein, Containers, and Cloud: From LXC to Docker to Kubernetes, IEEE Cloud Computing (Volume: 1, Issue: 3, Sept. 2014)
4. A. Javed, Linux Containers: An Emerging Cloud Technology.
5. Containers on Windows and Windows Server Containers, https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/, accessed April 2019
6. D. Merkel, Docker: Lightweight Linux Containers for Consistent Development and Deployment, Linux Journal, March 2014.
7. K. Seo1, H. Hwang1, I. Moon1, O. Kwon1, B. Kim, Performance Comparison Analysis of Linux Container and Virtual Machine for Building Cloud, Advanced Science and Technology Letters Vol.66 (Networking and Communication 2014), pp.105-111
8. D. Quigley, J. Sipek, C. P. Wright, E. Zadok, UnionFS: User- and Community-oriented Development of a Unification Filesystem, In Proceedings of the 2006 Linux Symposium
9. R. Dua, Vaibhav Kohli, S. Patil, S. Patil, Performance analysis of Union and CoW File Systems with Docker, International Conference on Computing, Analytics and Security Trends (CAST)
10. B, Burns and D. Oppenheimer, Design patterns for container-based distributed systems, The 8th Usenix Workshop on Hot Topics in Cloud Computing (HotCloud '16)
11. B. I. Ismail at al. Evaluation of Docker as Edge Computing Platform, 2015 IEEE Conference on Open Systems (ICOS), August 2015
12. Evaluation of Docker for IoT Application, Vel Tech Technical University, India, International Journal on Recent and Innovation Trends in Computing and Communication, Volume: 4 Issue: 6 ,pp. 624 – 628
13. Docker documentation, Docker images are multiplatform, https://blog.docker.com/2017/09/docker-official-images-now-multi-platform/, Accessed April 2019
14. Kubernetes official documentation, https://kubernetes.io/blog/2015/11/creating-a-raspberry-pi-cluster-running-kubernetes-the-shopping-list-part-1/, accessed April 2019.
15. A. R. Manu at al. Docker Container Security via Heuristics-Based Multilateral Security-Conceptual and Pragmatic Study, 2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT). March 2016
16. B. Burns, B. Grant, D. Oppenheimer, E. Brewer, J. Wilkes, Borg, Omega, and Kubernetes, ACM Queue, ppt. 70-93, vol. 14
17. A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune and J. Wilkes, Large-scale cluster management at Google with Borg, Proceedings of the European Conference on Computer Systems EuroSys 2015, Bordeaux, France
18. Unikernels, Russell Pavlicek, O'Reilly Media, Inc., October 2016, 9781492042815.
19. Madhavapeddy A., Mortier R., Rotsos C., Scott D., Singh B., Gazagnaire T., Smith S., Hand H. and Crowcroft J., ASPLOS '13 Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems, 461-472.
20. Unikernels joins Docker, https://blog.docker.com/2016/01/unikernel/, accessd april 2019