A Cloud-Agnostic Container Orchestrator for Improving Interoperability

David Elliott*, Carlos Otero*†, Matthew Ridley*, Xavier Merino†
*Northrop Grumman Corporation
Email: {david.elliott, matthew.ridley}@ngc.com
†Department of Electrical and Computer Engineering, Florida Institute of Technology
Email: cotero@fit.edu, xmerino2012@my.fit.edu

***Abstract*—The last several years have seen a rapid increase in the use of containers and containerizing services, such as Docker, Kubernetes, and LXC. With such a rise comes a vast increase in opportunities for improved portability, security, and automation. While many of these opportunities are being taken advantage of in containers today, there remains many more possibilities that have not been explored. In this paper, we present a novel approach to container management that enables the rapid live migration of stateful containers between any hosts within private, public, or hybrid clouds, through a simple interface that provides a high-level view of the network of computing nodes and the containers running on them. Finally, we provide a vision for container management that allows greatly improved security and resiliency through autonomous and self-migrating containers existing across the scope of the Internet.**

***Keywords*-cloud computing; containers; orchestrator; live migration; interoperability;**

I. Introduction

Since its inception, Linux containers have steadily gained traction in industry. According to Rightscale's 2018 annual survey of 997 IT professionals, with 53% of respondents from enterprises with over one thousand employees, 49% are using Docker containers in their business. Another 29% also plan to use it in 2018 [1]. Given that Docker containers had just 13% adoption in a similar survey in 2015 [2], it seems clear that industry has found great value in them. Among the many reasons for this rapid adoption is the increase in portability brought about by operating system virtualization. This allows containers to share system resources as if they were applications running on the host itself. Examples of container engines include Docker, Kubernetes, Containerd, Runc, and LXC, with Docker being the most popular and Kubernetes following close behind it [1].

A tool called Checkpoint/Restore In Userspace (CRIU) was created to allow processes to be checkpointed and restored in Linux. This tool can also be used for checkpointing and restoring Linux containers. CRIU can even be used to checkpoint a container on one machine and restore it on another through a process known as live migration.

Operating system virtualization allows almost any Linux container to be deployed and run on almost any Linux system. However, when performing live migration, there are many more dependencies that have to be accounted for when moving across systems and platforms. When a process runs, it uses certain kernel and CPU features that are specific to the host system that it is running on. Since containers run as processes on the host system, these kernel and CPU features must be accounted for when live migrating to a destination system.

A substantial problem encountered in industry is the lack of interoperability between clouds [3], which makes it difficult for users to take resources running in one cloud and transfer them to another. Linthicum [4] attributes this to the rapid scaling that cloud providers underwent early on, and the fact that portability is not in cloud providers' best interests. To solve this problem, we provide a novel approach to managing and migrating multi-container applications across cloud providers, through an adapter connecting hosts in the cloud to a centralized orchestration node, an orchestration node with a web interface for convenient and real-time access, and an abstraction that groups containers in order to perform operations on many at once.

II. Related Work

The work by Dillon *et al.* [5] addresses interoperability by noting concerns over vendor lock-in and the effect that an intermediary layer might have in ameliorating these issues. The authors also state the importance of major cloud vendors supporting a standard. Many others express a similar sentiment toward cloud providers supporting a standard [3], [6]–[9]. However, this has yet to occur, so it continues to be necessary to explore alternate solutions.

Early analysis of cloud interoperability by Parameswaran and Chaddha [7] found that two major approaches have emerged toward ensuring user needs are met: creation of a unified cloud interface (UCI) and creation of a cloud orchestration platform. A UCI is an API about APIs, in the sense that it provides an interface with which a user can engage, while under the hood it is interacting with the unique APIs of different cloud providers to execute a user's applications. The authors note that the UCI could be provided as a web service with a dashboard showing the state of allocated resources and running VMs. The cloud orchestration platform is similar in concept to the UCI, but with the major difference that cloud providers must register with the orchestrator. After registration, the orchestrator is free to execute jobs on any platform available to it, making choices based on cost, performance, and latency as necessary to optimize execution. We consider our work to be a type of orchestration platform, though we do not require cloud providers to register with it explicitly; instead, registration with our service is done via the adapter spoken about in Section III. Furthermore, we have taken the dashboard approach from the UCI and incorporated it into our system.

Arunkumar and Venkataraman [10] make a distinction between application, platform, storage, and management interoperability. They describe concerns in application interoperability to be apparent when an application needs re-engineering in order to move it to another platform or provider, due to having been bound to a specific provider's API. Platform interoperability is at risk when the platforms on which applications are running, such as virtual machines (VMs) or custom development platforms, are not able to be ported to another cloud, usually due to coupling with hardware. Storage interoperability addresses concerns relating to specific RAID configurations and partitioning that may have been done by a cloud provider or on local premises, which prevents data transference or results in corruption. Lastly, management interoperability refers to details of configuration, such as network adapters, open ports, and inter-process communication that may not have an equivalent on another provider's system and thus prevent migration. We address application interoperability through the use of containers and platform interoperability through the use of adapters. Storage interoperability is not a concern at the level of abstraction that we are operating, and management interoperability is addressed by both the adapter and the orchestrator in our system.

III. Approach

The approach for our system is shown in Figure 1. The system itself is made up of several parts: the orchestrator, adapters, hosts, applications, and containers. The orchestrator is a single node that has control over the entire system: it runs specialized software different from that of hosts, which listens for remote procedure calls (RPC) coming from hosts through adapters as they send statistics and command responses. The orchestrator also contains a management site to allow users to interact with and control the system. Hosts are computing devices configured to work with our system through an adapter. Containers run on hosts using Runc, a lower-level container runtime that provides close use of CRIU. This allows us to make use of certain CRIU options, such as TCP connection checkpointing, that container engines such as Docker do not yet support.

A. System Adapters

An adapter in our system is composed of a set of operations that are performed on a host, which establishes a connection with the orchestrator and supports live migration.

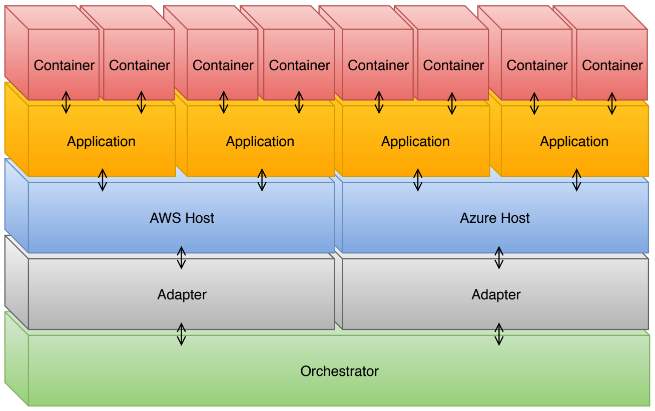

Figure 1. The system stack is based on an orchestrator that is able to connect to hosts through an adapter. Once connected, the orchestrator can deploy and run applications, or sets of containers, on the host. From there, the orchestrator can move applications between hosts via live migration.

For those cloud providers that support uploading custom VM's to run on their infrastructure, we can simply upload our pre-provisioned VM. In this case both the host and adapter are sent to the cloud in the form of a single VM. For all others, we launch a new VM and perform the same provisioning operations on that host. Provisioning requires installing various modules and packages required to support our system, including Runc and CRIU, as well as updating the kernel to match other hosts in the system. We currently perform this provisioning manually, but the process can easily be automated. As a result of this, we note that our method can be used to enable orchestration with any cloud provider that supports running and connecting to Linux virtual machines. Furthermore, although in the current state of the system we perform provisioning on new hosts that we launch, provisioning can be performed on existing hosts as well. Root permissions are currently required in order to configure a host with an adapter.

B. Live Migration

By providing adapters for all hosts in the system, we make it possible to deploy and migrate containers between them through the orchestrator. In order to do this, we perform a series of RPCs to execute the migration sequence. RPCs are sent throughout our system using RabbitMQ, an open source message broker. The sequence is as follows to migrate an application from host A to host B: (1) from the orchestrator, send an RPC to host A to indicate which application should be migrated and where, (2) host A checkpoints the application, (3) host A `rsync`'s the application's state to host B over SSH, (4) host A sends a response to the orchestrator to indicate that the transfer has finished, (5) the orchestrator sends an RPC to host B to indicate the application that needs restoring, (6) host B restores the application.

We note that there are several improvements that can be made to the migration process to reduce migration time and

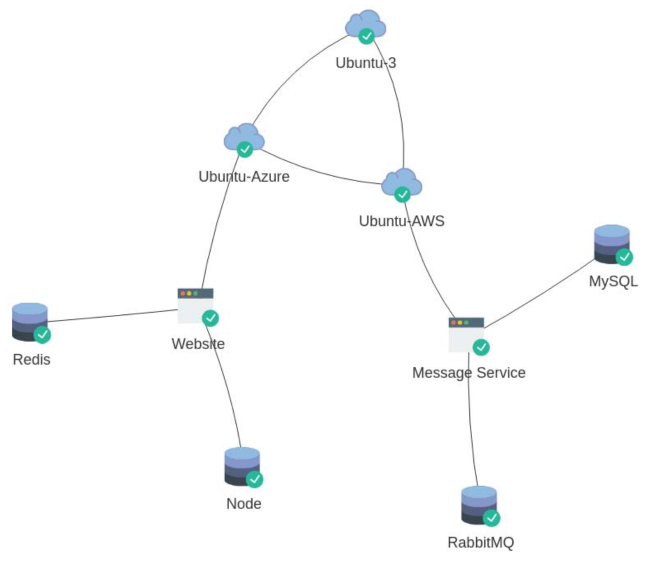

Figure 2. A live orchestrator view of a test configuration consisting of three hosts: one local to our laboratory (named Ubuntu-3), one in Azure (named Ubuntu-Azure), and one in AWS (named Ubuntu-AWS). The Azure and AWS hosts each have an application running on them that consists of two containers. All hosts are running Ubuntu 16.04.3 LTS with kernel version 4.13.

application downtime, such as memory pre-copy and post-copy, with further optimizations described in [11]; however, since the primary intent of this work is to demonstrate the improvement that our system provides to interoperability rather than improvements to live migration efficiency, these are left for a future work.

A limitation of container migration is that for all hosts that will take part in a migration process, CPU instruction sets must be similar. This limitation becomes relevant when attempting to interact with custom machines that may have older processors that are missing certain CPU flags, such as `xsave` [12]. In those cases, we identified two solutions, the first of which was to use QEMU (Quick Emulator) user emulation to dynamically translate processor instructions to a base set of instructions existing across CPUs. This had the advantage of allowing migration to a host with nearly any processor, but it introduced a performance decrease for the lifetime of the container. Alternatively, the kernel could be built with certain CPU capabilities, such as `xsave`, disabled. This had the advantage of ensuring that all containers ran at native speeds, but required additional configuration and time to set up the adapter. Since we already required a kernel modification on hosts in our system, we chose the second option.

C. Application Abstraction

Container applications often rely on multiple containers that depend on each other to function, such as with micro-service architectures. In order to facilitate the migration of multi-container services, our approach uses the concept of an application as the unit of operational work. That is, an application is made up of one or more containers, and when operated on, either through deployment on a host, stopping the application, or live migrating the application between hosts, all containers associated with the application are also operated on. This abstraction is to similar to abstractions made by other orchestration tools, such as the "pod" abstraction used by Kubernetes.

D. System Walkthrough

Using our system, a user would have a visual representation of their entire cloud infrastructure, including hosts, applications, and containers, as shown in Figure 2. This cloud infrastructure could be composed of systems from various cloud platforms such as Amazon Web Services, Microsoft Azure, Google Cloud, or local private servers. The user could then select an application to migrate, a destination host to migrate to, and the state/checkpoint from which to restore. The applications and all of its encapsulated containers would then be statefully migrated to the new host, even if it is a great distance away with vastly different performance capabilities, where it would continue running.

IV. Future Work

A natural extension of this work is to automate the process of deciding when and where to migrate. This is spoken about in a variety of contexts, such as moving target defense as an improved security measure [13], [14], decreasing latency in an edge-computing scenario [15]–[17], and optimizing cloud resources in factories of the future [18]. Our work supports the future vision of autonomous mission-critical systems, where containers statefully move across distributed systems to position themselves on targets that are better equipped to increase security. For example, autonomous decisions can be made using security posture metrics like the one presented in [19], where the measure by which movement is determined is based on security metrics derived from the system, rather than only based on performance, latency, and cost. However, metrics relating to security and resiliency are still in the early phases on development [19], and require further work before automation can be properly implemented. We find that it is primarily for this reason that an orchestrator is so vital at this time, as it provides a functional stepping-stone to more advanced, distributed orchestration.

Primary to the thread of research relating to applications as sets of containers is the potential for interconnected containers to be live-migrated to another host without losing connection to each other. If this could be completed, it could enable an entirely new container development paradigm, where large applications are broken up into containers that can be moved in whole from one host to the next. For example, a simple machine learning application may consist of two containers, one performing a long-running training algorithm, the other running a database with training data. With a paradigm like the one described here, it becomes possible to migrate the entire application, while training, to a pre-provisioned cloud instance in order to improve the training speed. This paradigm may limit applications to individual hosts initially, since simultaneously checkpointing containers across large distances will likely prove challenging. To solve these challenges, it may require collaboration with the developers of CRIU, and thus may rely on the willingness of current field experts in order to be brought to life. Yet, this possibility may greatly improve the usefulness of live migration and we believe is worth pursuing.

When restoring from a checkpoint, the file system of the container must be the same as it was when the checkpoint was made. Therefore, currently rolling back to previous checkpoints only works for containers whose file systems do not change. In order to fully support incremental checkpoints, checkpoints of the container file system must be made for each checkpoint of the memory state. This could be done through full standalone backups of the file system or through an incremental file system change tracker.

It is also possible, through this system, to support multiple architectures. This can be done through the previously mentioned QEMU user emulation, by either embedding QEMU within each container or by mounting QEMU at runtime. Since this system is designed to run containers without alterations, embedding QEMU is not possible. Therefore, it remains a future work to build our current system for other architectures, such as ARM, and configure it to work with QEMU to allow migration between architectures.

Acknowledgement

This work was supported by Northrop Grumman Corporation (NGC). The authors would like to thank the staff of the technology office at NGC - Mission Systems for their support.

References

[1] K. Weins, "Cloud computing trends: 2018 state of the cloud survey," Feb 2018. [Online]. Available: https://www.rightscale.com/blog/cloud-industry-insights/cloud-computing-trends-2018-state-cloud-survey

[2] ——, "Cloud computing trends: 2015 state of the cloud survey," Feb 2015. [Online]. Available: https://www.rightscale.com/blog/cloud-industry-insights/cloud-computing-trends-2015-state-cloud-survey

[3] R. Ranjan, "The cloud interoperability challenge," *IEEE Cloud Computing*, vol. 1, no. 2, pp. 20–24, 2014.

[4] D. S. Linthicum, "Moving to autonomous and self-migrating containers for cloud applications," *IEEE Cloud Computing*, vol. 3, no. 6, pp. 6–9, 2016.

[5] T. Dillon, C. Wu, and E. Chang, "Cloud computing: issues and challenges," in *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*. Ieee, 2010, pp. 27–33.

[6] R. Cohen, "Examining cloud compatibility, portability and interoperability," *ElasticVapor: Life in the Cloud, DOI: http://www.elasticvapor.com/2009/02/examining-cloudcompatibility.html*, 2009.

[7] A. Parameswaran and A. Chaddha, "Cloud interoperability and standardization," *SETlabs briefings*, vol. 7, no. 7, pp. 19–26, 2009.

[8] D. Petcu, "Portability and interoperability between clouds: challenges and case study," in *European Conference on a Service-Based Internet*. Springer, 2011, pp. 62–74.

[9] B. Varghese and R. Buyya, "Next generation cloud computing: New trends and research directions," *Future Generation Computer Systems*, vol. 79, pp. 849–861, 2018.

[10] G. Arunkumar and N. Venkataraman, "A novel approach to address interoperability concern in cloud computing," *Procedia Computer Science*, vol. 50, pp. 554–559, 2015.

[11] H. Nie, P. Li, H. Xu, L. Dong, J. Song, and R. Wang, "Research on optimized pre-copy algorithm of live container migration in cloud environment," in *International Symposium on Parallel Architecture, Algorithm and Programming*. Springer, 2017, pp. 554–565.

[12] P. Guide, "Intel® 64 and ia-32 architectures software developers manual," *Volume 3B: System programming Guide, Part*, vol. 2, 2011.

[13] M. Villarreal-Vasquez, B. Bhargava, P. Angin, N. Ahmed, D. Goodwin, K. Brin, and J. Kobes, "An mtd-based self-adaptive resilience approach for cloud systems," in *Cloud Computing (CLOUD), 2017 IEEE 10th International Conference on*. IEEE, 2017, pp. 723–726.

[14] M. Azab, B. M. Mokhtar, A. S. Abed, and M. Eltoweissy, "Smart moving target defense for linux container resiliency," in *Collaboration and Internet Computing (CIC), 2016 IEEE 2nd International Conference on*. IEEE, 2016, pp. 122–130.

[15] L. Ma, S. Yi, and Q. Li, "Efficient service handoff across edge servers via docker container migration," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. ACM, 2017, p. 11.

[16] C. Dupont, R. Giaffreda, and L. Capra, "Edge computing in iot context: horizontal and vertical linux container migration," in *Global Internet of Things Summit (GIoTS), 2017*. IEEE, 2017, pp. 1–4.

[17] K. Ha, Y. Abe, Z. Chen, W. Hu, B. Amos, P. Pillai, and M. Satyanarayanan, "Adaptive vm handoff across cloudlets," *Technical Report CMU-CS-15-113, CMU School of Computer Science*, 2015.

[18] T. Nodehi, R. Jardim-Goncalves, A. Zutshi, and A. Grilo, "Icif: an inter-cloud interoperability framework for computing resource cloud providers in factories of the future," *International Journal of Computer Integrated Manufacturing*, vol. 30, no. 1, pp. 147–157, 2017.

[19] M. Ridley, C. E. Otero, X. Merino, and D. Elliott, "Quantifying the security posture of mission-critical systems," *IEEE Southeast Conference*, 2018.