Autonomic Orchestration of Containers: Problem Definition and Research Challenges

Emiliano Casalicchio
Blekinge Institute of Technology
Karlskrona, Sweden
Emiliano.Casalicchio@bth.se

ABSTRACT

Today, a new technology is going to change the way cloud platforms are designed and managed. This technology is called container. A container is a software environment where to install an application or application component and all the library dependencies, the binaries, and a basic configuration needed to run the application. The container technology promises to solve many cloud application issues, for example the application portability problem and the virtual machine performance overhead problem. The cloud industry is adopting the container technology both for internal usage and as commercial offering. However, we are far away from the maturity stage and there are still many research challenges to be solved. One of them is container orchestration, that make it possible to define how to select, deploy, monitor, and dynamically control the configuration of multi-container packaged applications in the cloud. This paper presents the problem of autonomic container orchestration, analyze the state of the art solutions and discuss open challenges.

CCS Concepts

• **Computing methodologies~Self-organization** • **Computer systems organization~Cloud computing** • *Information systems~Computing platforms* • *Applied computing~Service-oriented architectures* • Theory of computation~Distributed computing models

Keywords

Container; Docker; Cloud Computing; Autonomic Computing; Service Orchestration.

1. INTRODUCTION

Operating system and application virtualization, also known as *container* (or *Docker container* or simply *Docker* [1]), became popular since 2013 with the launch of the Docker open source project (docker.com) and with the growing interest of PaaS providers [2] and Internet service providers [3]. A container is a software environment where one can install an application or application component (the so called microservice) and all the library dependencies, the binaries, and a basic configuration needed to run the application. Containers provide a higher level of abstraction for the process lifecycle management, with the ability not only to start/stop but also to upgrade and release a new version of a containerized service in a seamless way. IBM researcher *"believe that the new cloud operating environment will be built using containers as a foundation. No virtual machines…"*[4].

Containers became so popular because they potentially may solve many cloud application issues, for example: the "*dependency hell" problem*, typical of complex distributed applications. Containers give the possibility to separate components, wrapping up the application with all its dependencies in a self-contained piece of software, that can be executed on any platform that supports the container technology. The *application portability problem*; a microservice can be executed on any platform supporting container and moreover the Docker container management framework is compliant with the cloud portability frameworks TOSCA [5][6]. The *performance overhead problem*; containers are lightweight and introduce lower overhead compared to VMs. The literature shows that the overhead introduced by containers, compared to bare-metal installations, is around 10% while hardware virtualization (classical VMs) introduce an overhead of the 40% [7]; moreover, launching and shouting down a container requires 1-2 seconds rather than minutes. Besides, the concept of microservices and the portability feature of containers make it possible to satisfy typical constraints imposed by the legislation and the regulation e.g., the data sovereignty and the vendor-lock-in.

For all those reasons, and more, the cloud industry adopted the container technology both for internal usage [8] and for offering container-based services and container development platforms [2]. Examples are Google container engine [8], Amazon ECS [9], Alauda (alauda.io), Seastar (seastar.io), Tutum (tutum.com), Azure Container Service (azure.microsoft.com). Containers are also the state of the art solution adopted to deploy large scale applications, for example big data applications requiring high elasticity in managing a very large amount of concurrent components (e.g. [10][11]).

Despite that wide interest in containers, we are far away from the maturity stage and there are still many challenges to be solved, for example: the need for reducing networking overheads compared to hypervisors; the need for secure resource sharing and isolation to enable multi-tenancy; the need for improving container monitoring capabilities; the need for improving container resource management at run time (e.g. vertical scaling of a container is not possible) and the need for improving orchestration policies and adaptation models adding autonomic capabilities.

This work defines and discusses the problem of container orchestration with a focus on autonomic mechanisms. The paper is organized as in what follow. Section 2 defines container orchestration. Section 3 discusses the related work. The research problem of autonomic container orchestration is formulated in Section 4. Research challenges and final remarks are presented in Section 5.

Conference'10, Month 1–2, 2010, City, State, Country.

DOI: http://dx.doi.org/10.1145/12345.67890

2. CONTAINER ORCHESTRATION

Orchestration is intended as the set of operations that cloud providers and application owners undertake (either manually or automatically) for selecting, deploying, monitoring, and dynamically controlling the configuration of resources to guarantee a certain level of quality of service [6]. In the same way, *container orchestration allows cloud and application providers to define how to select, to deploy, to monitor, and to dynamically control the configuration of multi-container packaged applications in the cloud.*

Container orchestration is not only concerned with the initial deployment of multi-container applications, but it also includes the management at runtime, for example: scaling a multi-container application as a single entity; controlling multi-tenant container-based applications; composing containers in high available software architectures; optimizing the networking of the application, e.g. executing computation near the data.

Because the highly dynamic and complex environment in which containers are operated, the orchestration actions must be automatically adapted at run time, without human intervention, and that entails autonomic orchestration mechanisms.

Today, the container orchestration frameworks are at their infancy and do not include any autonomic feature [12]. Cloudify (getcloudify.org) and Kubernetes (kubernetes.io) are the main TOSCA compliant implementations allowing orchestration of Docker containers. However, how to execute and orchestrate container in a distributed environment without leveraging on hypervisors is still an open issue. Docker Swarm requires a static setup of the cluster nodes. Core OS (coreos.com) is a first step toward in this direction, but it is a young solution, and again it is effected by the standardization problem. The auto-scaling policy implemented in Kubernetes is a simple threshold based algorithms that only uses CPU usage metrics. The scheduling and load distribution policy implemented in Swarm is rather naïve.

To realize autonomic container orchestration many research problems must be addressed, among them: resource management at run time, synchronization between design and execution environments, monitoring, performance profiling and characterization, performance modeling, definition of optimization models for orchestration and adaptation.

3. RELATED WORK

Autonomic orchestration of Docker containers could leverage more than ten years of research results in the field of autonomic computing [15], autonomic service oriented systems [16], autonomic cloud computing [17][18].

A recent survey on service level management in the cloud [17] found that: the monitor-analysis-plan-execute (MAPE) architecture style [14] is predominant in autonomic cloud systems; and, heuristics (e.g. [19][13]), and optimization policies (e.g. [20][13][21]) are the most commonly used techniques for resource allocation. In [18] the authors review literature on autonomic QoS-aware resource management in cloud computing since 2009. Both from [17] and [18] it results that mainly researchers consider time, availability, cost (in general), energy consumption, as main constraints or objective of the adaptation. A few works contemplate security, but no-one considers constraints imposed by legal rules [22][23], e.g. sovereignty of data.

The idea of container dates back to 1992 [24] and matured over the years with the introduction of Linux namespace [25] and the LXC project (linuxcontainers.org), a solution designed to execute full operating system images in containers. From operating system virtualization the idea has moved to application container [1]. The experiments in [7] show that Docker performs as bare-metal systems and outperforms KVM in most of the cases but that it is weak for I/O and network intensive workloads. In [26] the authors shown that for specific memory configuration Docker is capable to perform slightly better than VMs while executing scientific workloads.

The adoption of container technologies calls for new autonomic management solutions. An early study on container management is presented in [27]. Here the authors compare a VM-based and Elastic Application Container based resource management with regards to their feasibility and resource-efficiency. The results show that the container-based solution outperforms the VM-based approach in terms of feasibility and resource-efficiency. Today, there are technologies that specifically support container orchestration but the level of automation is very naïve. CoreOS integrates an orchestrator, called fleet, that supports live container migration from one host to another and the possibility to modify at runtime the environment of running containers. Google Kubernetes adopts a mixed approach to the virtualization that allows to scale both the computing resources available to containers and the number containers available to applications. In that way, application can rapidly scale up and down according to their needs. In [12] the authors analyze the state of the art and challenges in container orchestration pointing out the high fragmentation of technologies, lack of standard image format and immaturity of monitoring systems. C-Port [28] is the first example of orchestrator that make it possible to deploy and manage container across multiple clouds. The authors plan to address the issues of resource discovery and coordination, container scheduling and placement, and dynamic adaptation. However, the research is at an early stage. In term of orchestration policy, they developed a constraint-programming model that can be used for dynamic resource discovery and selection. The constraints that they considered are availability, cost, performance, security, or power consumption.

4. RESEARCH PROBLEM

The orchestration of container is a broad research topic because it covers the whole life-cycle management. In this work we focuses on the elastic properties of container management over distributed datacenters and in multitenant environments. We concentrate our attention on **why**, **when**, **where** and **how** to deploy, to launch, to run, to migrate and to shut down container. All those actions are defined in what is called an **orchestration** or **adaptation policy**. **Why** orchestration action should occur is determined by specific events such as an increase in the workload intensity or volume, the arrival of a new tenant, change of SLA, node failures, critical node load/health state, and so on. Figure 1 shows our approach to autonomic container orchestration. It is based on the classical MAPE-K cycle [14].

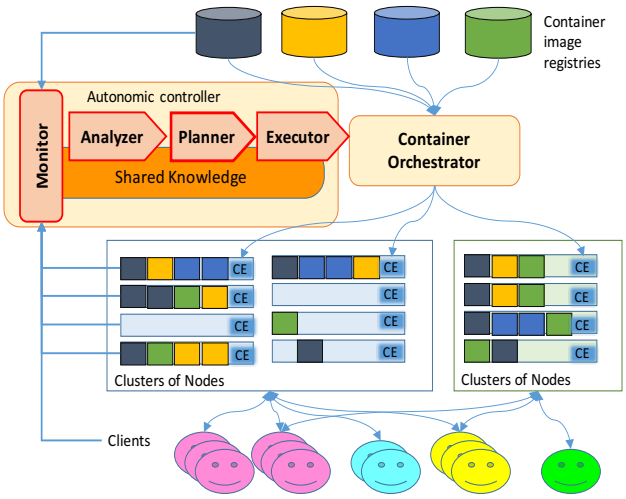

Figure 2 The container orchestration approach

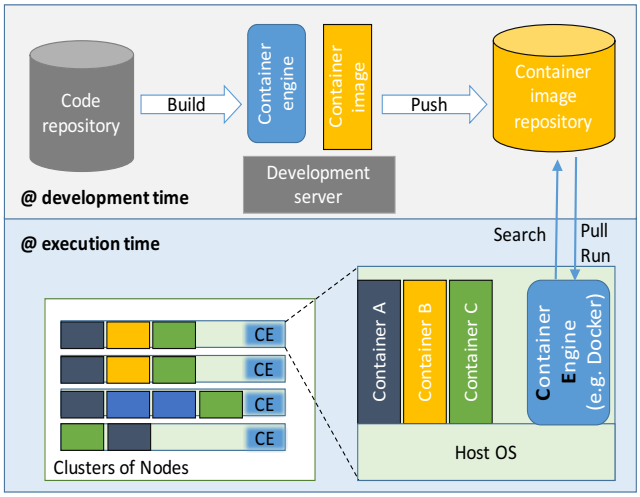

Figure 1 Container life cycle

Containers are stored in repositories called Container Image Registries. The orchestrator decides also **when** to adapt the system (e.g. to start the execution of a container) and **where** the adaptation will take place (e.g. on what computing node containers will run). Computing nodes are grouped in clusters that can be part of the same datacenter or can be geographically distributed. Each computing node in a cluster can run one or more containers, and the execution of containers is managed by the container engine (CE) on the node (see Figure 2). The orchestrator communicates with the container engine of each node to coordinate the execution of the containers. The decisions taken by the orchestrator are based on the system and environment state information collected by the Monitor component and analyzed by the Analyzer component (see Figure 1). The Planner component executes the orchestration policy to find the appropriate system configuration, that is, it solves a system model, or it runs an heuristic algorithm. Typically, the adaptation policy finds the optimal or sub-optimal system configuration that maximizes the provider utility and that satisfies non-functional and functional constraints. The Executor component implements the adaptation plan properly interacting with the Container Orchestrator component (e.g., Kubernetes). **How** the container is deployed and executed is another open issue. In the problem description we envisioned a centralized approach to orchestration. However, also decentralized solutions, i.e. choreography, apply and must be carefully investigated.

5. RESEARCH CHALLENGES

As pointed out in the previous sections, state of the art mechanisms for container orchestration should be enhanced introducing models and algorithms for runtime self-adaptation. Among the many research challenges, an urgent answer is required for the followings.

Monitoring, profiling and characterization. Monitoring of containers includes monitoring of the containerized environment (i.e., of the application) and monitoring of the container engine/platform. Monitoring techniques and tools used for the operating system and application levels do not allow to catch a wide range of QoS metrics and health state metrics for containers. Moreover, there is no a commonly agreed definition of QoS metrics for container based systems. Docker offers the `docker stat` command that returns CPU and memory utilization for each running container. More detailed CPU, memory and network statistics can be accessed through the `/containers/(id)/stats` API (docs.docker.com).

In a recent work [29], the authors the authors modify Docker and Docker Swarm in order to monitor the I/O capacity and utilization of the containers with the goal of controlling the QoS level of a Docker cluster.

Also performance characterization studies are limited, an example is [7]. Profiling and characterization of the performance of containers and container orchestration technologies is fundamental for the definition of performance models and autonomic orchestration policies.

Performance models. Validated performance models and energy consumption models of container-based systems are inexistent. Performance models are widely used in autonomic computing as representation of the system that must be adapted and as a tool to determine the reconfiguration actions needed to maintain the desired level of service. An alternative approach is the use of machine learning techniques to determine the more appropriate reconfiguration action (e.g. reinforcement learning [30]).

Adaptation models for container orchestration. As already pointed out, the container orchestration policies used until now are very simple and no autonomic mechanisms are used. This is also consequence of the lack of performance characterization and performance models. What the industry need is the definition of a framework for QoS-aware, energy-aware and legislation-aware optimal adaptation for container orchestration. This framework should allow to define system models, QoS, energy and legal constraints, to find optimal adaptation policies for container orchestration at run time.

That models, to be effective, must account for real systems limitations such as: the impossibility to change the resource assigned to a container at run time (that makes impossible vertical scaling) and the practical difficulty to synchronize the design and execution environment [31]. For example, if the execution environment is modified at run time for orchestration needs, that will impact the existing system architecture design, that must be synchronized. Off course, with the time, advancements in resource management and synchronization will overcome these limitation, and new adaptation policies can be designed.

To conclude, the way toward the next generation of cloud computing platforms is based on application virtualization rather

than on hardware virtualization; and it requires a strong contribution from the research community not only focused on the above discussed orchestration problem but also on different area, for example: improving networking management to reduce overheads compared to hypervisors networking; enhancing, secure resource sharing and isolation to enable multi-tenancy; refining application design methodologies to find the right balance between the micro-service fragmentation and performances.

6. ACKNOWLEDGMENTS

This work is funded by the research project "Scalable resource-efficient systems for big data analytics" financed by the Knowledge Foundation (grant: 20140032) in Sweden.

7. REFERENCES

[1] D. Merkel. (2014). Docker: lightweight Linux containers for consistent development and deployment. *Linux J.* 2014,

[2] R. Dua, A. R. Raja, D. Kakadia, (2014) Virtualization vs containerization to support PaaS, in: Proc. of 2014 IEEE Int'l Conf. on Cloud Engineering, IC2E'14, 2014, pp. 610 - 614.

[3] S. Natarajan, A. Ghanwani, D. Krishnaswamy, R. Krishnan, P. Willis and A. Chaudhary: An Analysis of Container-based Platforms for NFV. IETF draft, Apr. 2016.

[4] IBM Container Cloud Operating System project, http://researcher.watson.ibm.com/researcher/view_group.php?id=6302

[5] OASIS, (2013) Topology and orchestration specification for cloud applications, Tech. Rep. Ver.1.0, OASIS Standard

[6] B. D. Martino, G. Cretella, A. Esposito, (2015) Advances in applications portability and services interoperability among multiple clouds, IEEE Cloud Computing 2 (2) (2015) 22-28.

[7] W. Felter, A. Ferreira, R. Rajamony, J. Rubio, (2014) An Updated Performance Comparison of Virtual Machines and Linux Containers, IBM Research Report, RC25482 (AUS1407-001) July 21, 2014

[8] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes. (2016). Borg, Omega, and Kubernetes. *Queue* 14, 1, pages 10 (January 2016), 24 pages.

[9] W. Vogels, Under the Hood of Amazon EC2 Container Service, 20 July 2015, http://www.allthingsdistributed.com/

[10] W. Gerlach, et al.. (2014). Skyport: container-based execution environment management for multi-cloud scientific workflows. IEEE DataCloud '14. IEEE Press, Piscataway, NJ, USA, 25-32.

[11] DT-T. Nguyen et al. (2016). An Index Scheme for Similarity Search on Cloud Computing using MapReduce over Docker Container. In *ACM* IMCOM '16. ACM, New York, NY, USA, Article 60 , 6 pages.

[12] A. Tosatto, P. Ruiu, A. Attanasio,(2015) Container-based orchestration in cloud: State of the art and challenges, in: Proc. of 9th Int'l Conf. on Complex, Intelligent, and Software Intensive Systems, CISIS '15, 2015, pp. 70-75.

[13] E. Casalicchio, D.A.Menasce, A. Aldhalaan (2013), Autonomic resource provisioning in cloud systems with availability goals, ACM Conference on Autonomic Computing CAC'13, August 5–9, 2013, Miami, FL, USA

[14] J. O. Kephart, D. M. Chess, (2003) The vision of autonomic computing, in Computer, vol. 36, no. 1, pp. 41-50, Jan 2003.

[15] M. C. Huebscher and J. A. McCann. (2008). A survey of autonomic computing—degrees, models, and applications. *ACM Comput. Surv.* 40, 3, 7 (August 2008).

[16] A. L. Lemos, F. Daniel, and B. Benatallah. (2015). Web Service Composition: A Survey of Techniques and Tools. *ACM Comput. Surv.* 48, 3, Article 33 (December 2015).

[17] F. Faniyi and R. Bahsoon. (2015). A Systematic Review of Service Level Management in the Cloud. *ACM Comput. Surv.* 48, 3, Article 43 (December 2015), 27 pages.

[18] S. Singh and I. Chana. (2015). QoS-Aware Autonomic Resource Management in Cloud Computing: A Systematic Review. ACM Comput. Surv. 48, 3, Article 42 (December 2015), 46 pages

[19] A. Beloglazov, J. Abawajy, and R. Buyya. (2012). Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Generation Comput. Syst. 28, 5 (2012), 755–768.

[20] D. Ardagna, B. Panicucci, M. Trubian, and L. Zhang. (2012). Energy-aware autonomic resource allocation in multitier virtualized environments. IEEE Trans. Services Comput. 5, 1 (2012),

[21] M. Maggio et al. (2012). Comparison of decision-making strategies for self- optimization in autonomic computing systems. ACM Trans. Auton. Adapt. Syst. 7, 4 (2012) 1–32.

[22] E. Casalicchio, M. Palmirani (2015) A Cloud Service Broker with Legal-Rule Compliance Checking and Quality Assurance Capabilities, Procedia Computer Science, Elsevier

[23] B. D. Martino, G. Cretella, A. Esposito (2015) Towards a legislation-aware cloud computing framework, Procedia Computer Science (2015)

[24] R. Pike, D. Presotto, K. Thompson, H. Trickey, P. Winterbottom, (1993) The use of name spaces in plan 9, SIGOPS Oper. Syst. Rev. 27 (2) (1993) 72-76.

[25] E. W. Biederman, Multiple instances of the global Linux namespaces, in: 2006 Ottawa Linux Symposium, 2006.

[26] T. Adufu, J. Choi and Y. Kim (2015) Is container-based technology a winner for high performance scientific applications?, *Network Operations and Management Symposium, 17th Asia-Pacific*, Busan, 2015, pp. 507-510.

[27] S. He, et al. (2012) Elastic Application Container: A Lightweight Approach for Cloud Resource Provisioning, *IEEE 26th Int. Conf. on Advanced Information Networking and Applications*, Fukuoka, 2012

[28] M. Abdelbaky et al., (2015) Docker Containers across Multiple Clouds and Data Centers, *2015 IEEE/ACM 8th Int. Conf. on Utility and Cloud Computing*, Limassol, 2015.

[29] S. McDaniel, S. Herbein and M. Taufer, (2015) A Two-Tiered Approach to I/O Quality of Service in Docker Containers *2015 IEEE Int. Conf. on Cluster Computing*.

[30] A. Pelaez, A. Quiroz and M. Parashar, "Dynamic Adaptation of Policies Using Machine Learning," *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, Cartagena, Colombia, 2016,

[31] F. Paraiso, S. Challita, Y. Al-Dhuraibi, P. Merle. Model-Driven Management of Docker Containers. 9th IEEE International Conference on Cloud Computing (CLOUD), Jun 2016, San Francisco, United Stat