Optimizing Data Center Resource Management: A Comparative Study of Virtual Machine and Container Orchestration Tools

Daniel Thomas
daniel.cromwel@gmail.com

***Abstract*—As cloud computing evolves, data center administrators increasingly rely on orchestration tools to manage the complexities of resource provisioning, application deployment, and system scaling. This paper presents a comparative analysis of orchestration tools designed for both virtual machines (VMs) and containers, which are the two predominant technologies for running applications in modern data centers. We explore the core functionalities, advantages, and limitations of widely used orchestration platforms such as OpenStack, Chef, Ansible, Docker Swarm, Kubernetes, and Mesos. Focusing on container orchestration, which has gained prominence for its lightweight nature, we discuss the operational benefits of reduced overhead, faster deployment, and scalability. We also highlight the challenges in ensuring high availability, fault tolerance, and network resilience in distributed systems using these tools. Furthermore, this study evaluates the trade-offs between VMs and containers in terms of performance, isolation, and resource utilization, offering guidance for data center managers and software engineers in selecting the optimal orchestration platform. The paper concludes with recommendations for future developments in orchestration technologies that can further optimize cloud infrastructure management.**

I. Introduction

With the advent and proliferation of virtualization, computing as a service has taken off. There are now multiple cloud service providers, such as Google Cloud, Amazon Web Service, and Microsoft Azure. The operators of these cloud services typically have their own data centers, each consisting of thousands of physical machines connected via a high-speed and very low latency network such as Infiniband (IB) [1]. Data center operators use various tools to automate the process of management, provisioning and scaling the resources for their customers.

Orchestration tools have been developed to ease the operation of a data center. Tools such as OpenStack's Heat have been used to manage Linux kernel virtual machines [2]. There are many other tools, such as Chef and Ansible, for virtual machines [3], [4]. Container Orchestration tools such as Docker Swarm, Kubernetes, and Mesos have been developed to manage containers on mostly Linux machines [5]–[7]. In this survey, however, we shall focus more on the orchestration of containers than virtual machines.

Using these orchestration tools, data center operators can boot up virtual machines or containers on the physical machines, control the entire data center as a single entity, provision the necessary resources, and deploy and manage applications. These orchestration tools can also be used to automate the replication of processes, fault tolerance, failure detection and recovery, live migration of processes or entire virtual machines between physical machines, and also set up coordination between the processes to make the development of applications for the distributed system easier [8], [9]. Some orchestration tools also enforce process quotas, ensure Service-level agreement compliance, and monitor the data center's overall resource utilization and power consumption.

In this survey, the performance of these orchestration tools shall be compared. Also, these orchestration tools may offer different guarantees of tolerance for failures or network partitions. According to the CAP theorem, only two out of three: the presence of network partitions, availability, and consistency can be guaranteed by any distributed system [10]. In this case, the orchestration tools may employ different methods for error recovery and maintaining high reliability, offering various combinations of two of the three things that applications can easily support.

In the next section, we shall see some of the past work on virtualization, the use of virtual machines and containers, and the approaches taken by the different virtualization technologies to enforce isolation between applications. In Section 3, we shall describe how virtual machines and containers are controlled by the various orchestration tools. In Section 4, we analyze the limitations, evaluate the performance and ease of use, and also compare the different approaches they take

for coordination and failure. In Section 5, we discuss possible future work in the field and conclude in Section 6.

II. Background

The increasing proliferation and popularity of virtualization technology were necessary for the data center, and cloud service providers to thrive. Virtual machines allowed for running more isolated virtual operating systems or applications on a lesser number of physical machines. The use of these virtual machines could be leased out to third parties, providing a revenue stream for people who may have powerful physical machines but are unable to push them to their full potential. Consequently, cloud computing has made the concept of computing as a utility possible [12]. In this context, virtualization involves creating virtual hardware and running actual operating systems and applications on top of it. Each of these virtual machines is isolated and independent from one another and is managed by a hypervisor that is running on the physical hardware or the host operating system [9].

Virtualization technology is now decades old. Every major operating system that runs on servers has built-in virtualization support without the need for a third-party hypervisor. For example, the Linux kernel has KVM and Microsoft Windows Server has Hyper-V. Both also have well-documented endpoints for various front-ends to interface with, providing data center operators and server administrators with many tools to automate and manage their machines.

A similar development has been going on with containers on these operating systems as well. Like Control Groups (cgroups) in the Linux kernel [13], Hypervisor.Framework [14] in Apple macOS and Hyper-V in Microsoft Windows [15]. Container engines that provide a cross-platform frontend but make use of the respective container support structure in the host operating system have been developed, of which the most prominent is Docker [16].

Virtual machines are complete hardware virtualizations, with a full-fledged guest operating system on which applications are run. They enable the use of legacy applications that require legacy operating systems that only run on legacy hardware. The necessary legacy hardware can be emulated by the hypervisor, allowing the unmodified operating systems and applications to be run. There is also paravirtualization where a significant increase in performance can be achieved with some changes to the guest operating system, but it still does not entail modifying the application [9].

However, containers provide the isolation of virtual machines without much of the overhead. There is no guest operating system, and the host kernel itself enforces isolation [11]. However, this would also require the applications to be able to execute directly on the host operating system. Hence, this technique may not be used to run legacy applications.

Figure 1 shows the differences between virtual machines and containers [11]. Containers, with none of the overhead of virtual machines and the lack of guest operating systems, can be used to start up many orders of magnitude number of applications on the same host kernel. Containers are still new, however, and the implementation in the various operating system kernels may have bugs that compromise this isolation.

Orchestration tools for managing virtual machines and containers on many physical machines in a cluster have been developed. Tools for virtual machines that we shall evaluate in this survey include OpenStack [2], Chef [3] and Ansible [4]. Some of the tools for containers we shall cover in this survey are Docker Swarm [5], Kubernetes [**?**], and Mesos [7].

III. General Comparison

As we saw in the previous section, virtual machines are full-fledged operating systems, each containing its own set of libraries and running applications. These virtual machines are all isolated and are managed by the hypervisor running on the corresponding physical machine. Orchestration tools such as OpenStack Heat [2] or Ansible [4] or Chef [3] can be used to manage all the hypervisors of a cluster of physical machines, thereby controlling the entire data center as one entity.

However, containers have one key advantage that has greatly increased their popularity: significantly lower overhead compared to virtual machines. This is especially true when two or more applications link with different versions of libraries, but the operating system provides shared libraries that are a single version as depicted in Figure 2 [17]. You can run the applications in different contexts by running each one in its own virtual machine, but virtualization has a high overhead. Containers allow the kernel-level isolation of applications **and their libraries**. Figure 3 illustrates the use of containers for this use case [17].

Therefore, data center operators use a combination of virtual machines and containers on their physical clusters. Hence, orchestration tools for both are used in most data centers.

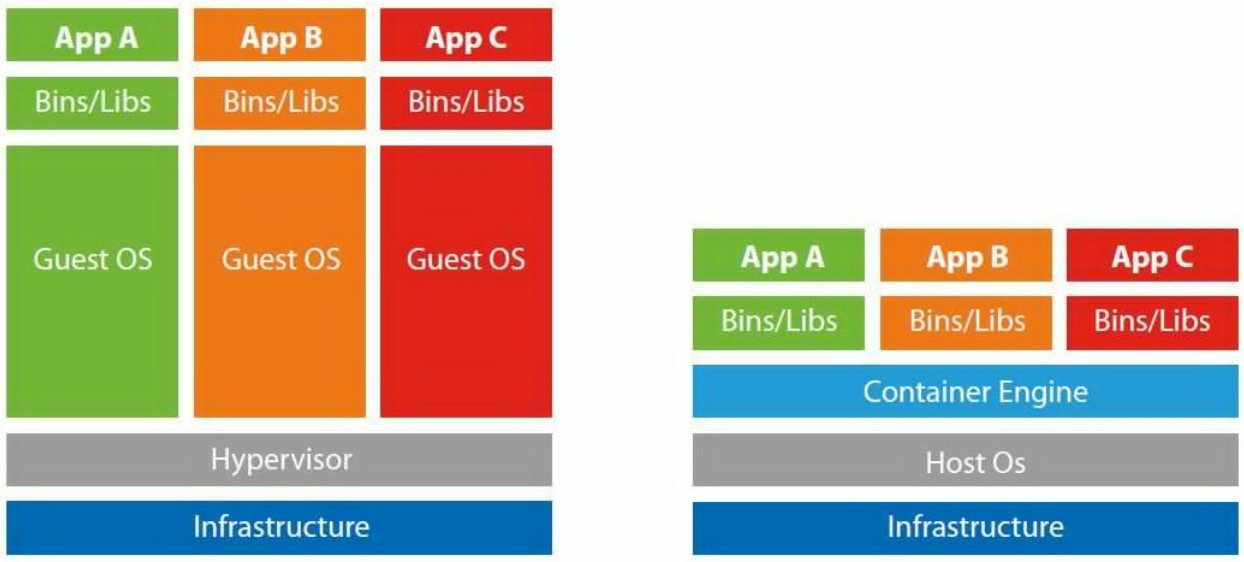

Fig. 1. Differences between Virtual Machines and Containers [11]

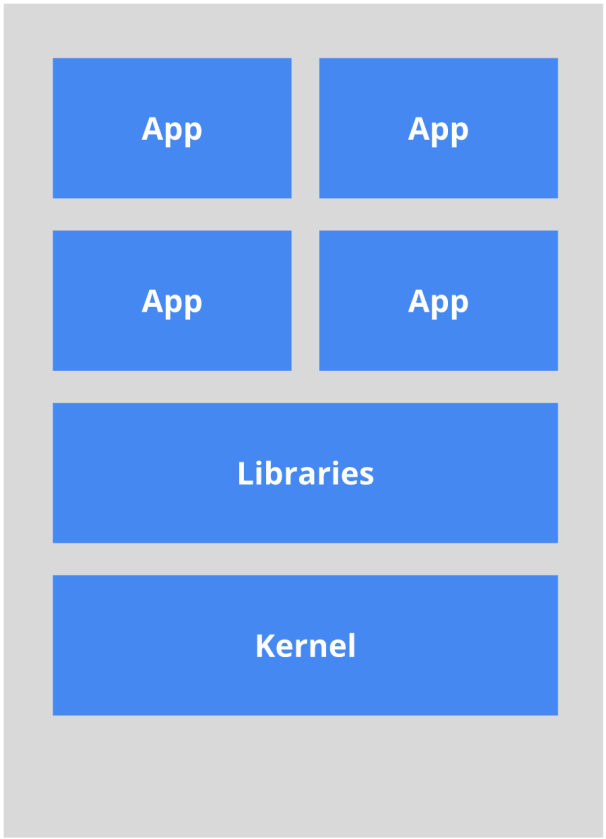

Fig. 2. Applications running natively on OS with shared libraries [17]

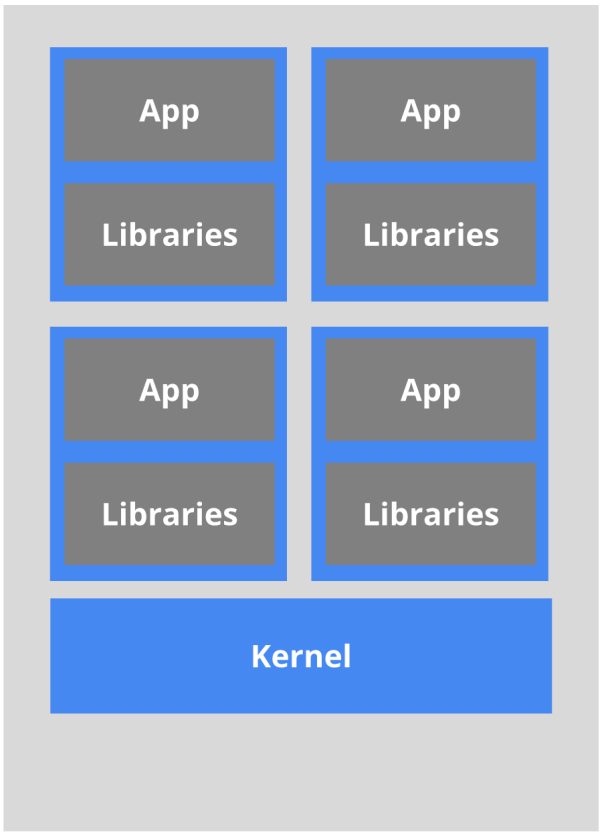

Fig. 3. Containers of applications and libraries running on OS [17]

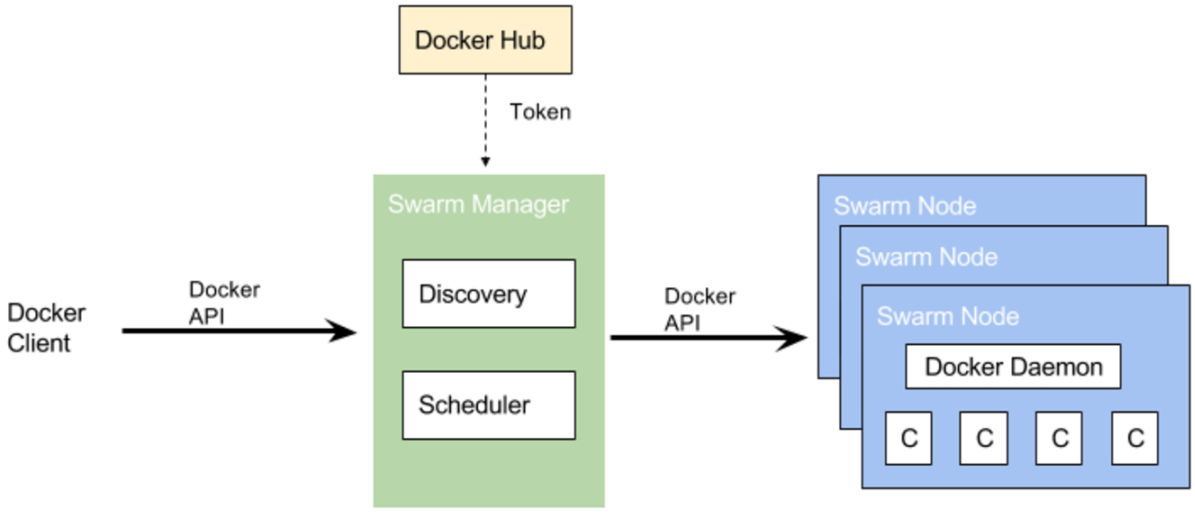

Fig. 4. Overview of Docker Swarm [18]

A. Virtual Machine Orchestration

As mentioned previously in this paper, the use of virtual machines is still necessary when a full-fledged guest operating system is needed. This is particularly the case when running legacy applications that require legacy operating systems and legacy hardware. However, advances in automation and orchestration can be applied to machines that emulate legacy hardware as well.

Ansible is an automation tool for managing massive physical clusters and deploying applications on them [4]. However, Ansible does not control the native operating system running on the physical server, but the virtual machines and applications running on it. It is made up of many components, not all relevant to this survey. We shall focus on the orchestration capabilities of Ansible. An administrator can execute hundreds or thousands of virtual machines, specifying an operating system and application image and setting up a complex network configuration between them. The administrator predetermines the configuration, which is written into the config file(s) and specified when running the orchestration tool.

Chef works similarly to Ansible [3] and orchestrates only the virtual machines and applications. Like Ansible, Chef also follows the master-slave architecture as depicted in Figure 5 [19]. Chef server contains the information about the Chef clients, which are the other nodes in the cluster. The Chef clients pull the specified configuration from the server and set up the nodes in the cluster. Application developers can use the Chef Developer Kit to quickly design and build the configuration and applications. This configuration is then pushed to the Chef server.

OpenStack Heat is another orchestration tool for managing bare physical servers [2]. Administrators can describe their cluster and specify the applications that are desired to run using a declarative language. It can be used in conjunction with Ansible or Chef to control the entire data center infrastructure.

B. Container Orchestration

We shall compare Docker Swarm [5], Kubernetes [**?**] and Apache Mesos [7]. Docker Swarm, Kubernetes, and Mesos all have a master-slave architecture, where a small number of manager nodes monitor the rest of the cluster [**?**], [5], [7].

Docker Swarm is an extension of the Docker container engine itself and is trivial to set up. The Docker Container Engine must be installed on the cluster nodes, each of which must also be associated with a discovery service. The discovery service backend is what the swarm manager(and their replicas) and the nodes use to authenticate themselves as members of the swarm cluster. The swarm manager is the one that controls the other nodes of the cluster. Replicas of the swarm manager take over in the case of a failure. [5]. Third-party tools that maintain the high availability of the swarm manager can be integrated into this setup [18].

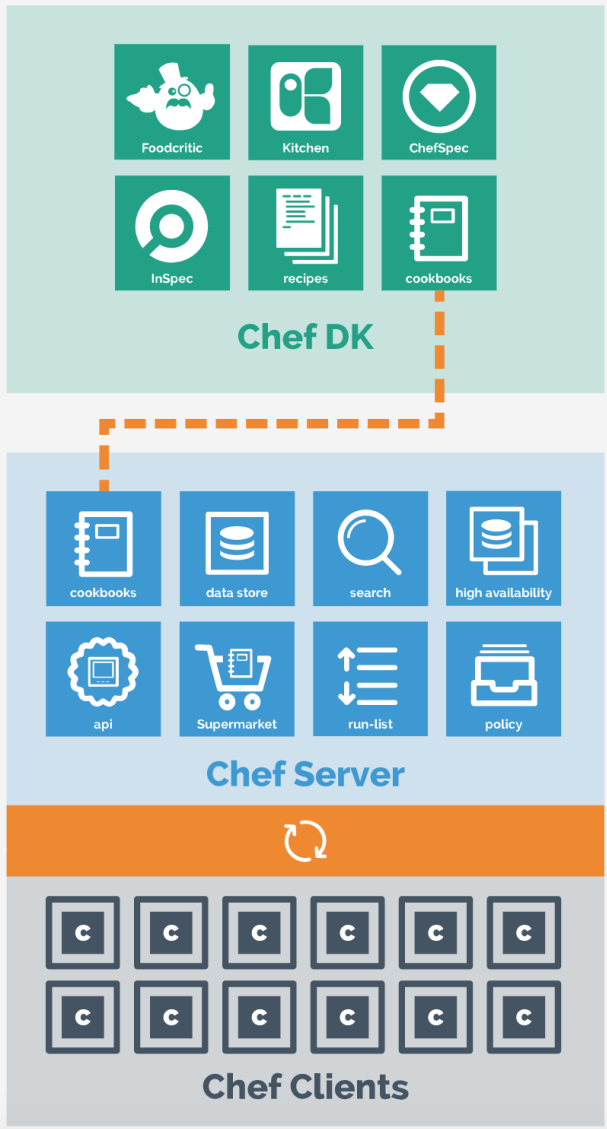

Fig. 5. Overview of Chef [19]

One key advantage of Docker Swarm is that it can be interfaced with using the same Docker API. As depicted in Figure 4, the same Docker API that is used by so many frontends is the same one used to control Docker Swarm [18].

Kubernetes is an orchestration tool developed by engineers at Google. Like Docker Swarm, Kubernetes also works with the Docker Container Engine [?]. It was originally for internal use only and was designed to orchestrate the use of their internal cluster, Borg [20]. But with the advances in containerization and the increasing use of containers to isolate applications, Google gave away some of the source code behind Kubernetes [21]. As illustrated in Figure 6, Kubernetes has a master-slave architecture [18]. Pods are groups of containers that run on the nodes in the cluster [?].

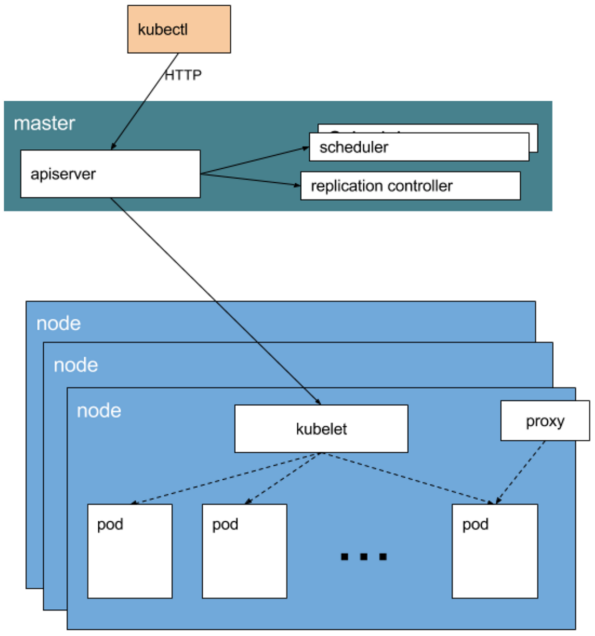

Fig. 6. Overview of Kubernetes [18]

Mesos is another orchestration tool that manages containers [7]. Mesos, like Docker Swarm and Kubernetes, allow data center operators to control the entire data center as a single entity. As depicted in Figure 7, Mesos has a master-slave architecture, like Docker Swarm and Kubernetes. The Mesos kernel runs on every node in the cluster. This enables the interfacing between the Mesos slave nodes and the Mesos master node. Using a coordination tool, like Zookeeper [22], the availability of the mesos master can be maintained by switching over to the master replicas when a master failure occurs. Mesos has frameworks for Hadoop MapReduce [23] and MPI; that application developers can use to build and run their Hadoop or MPI jobs using Mesos [7].

IV. Evaluations

Orchestration tools are vital for Developer Operations (DevOps). However, there is a real dearth of scholarly research work on the orchestration tools mentioned previously here. Therefore, we have also referenced whitepapers; documentation provided to data center operators, and performance and scalability resulting from

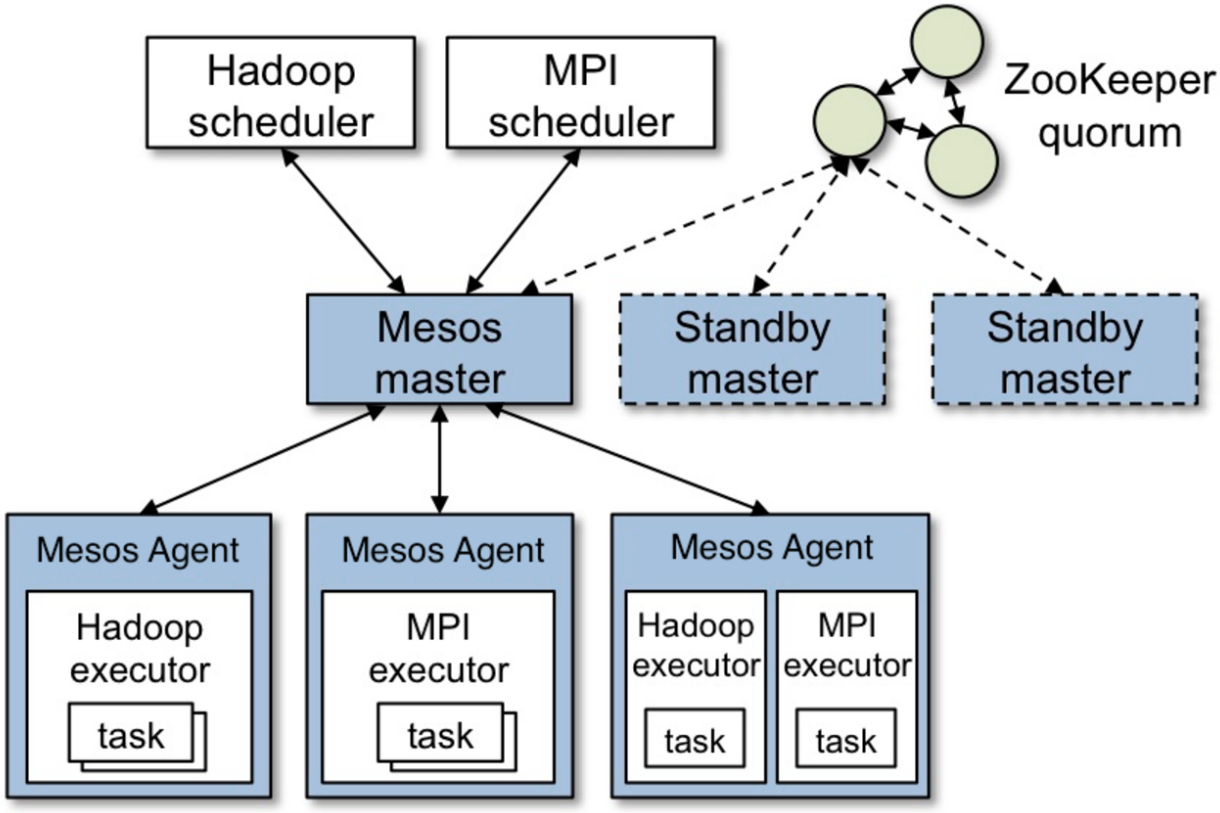

Fig. 7. Mesos Architecture [7]

benchmarking organizations [24]. As we have already seen, Orchestration tools, for the most part, are quite similar in nature. All of them attempt to solve some of the same kinds of problems, and some also in similar or even identical manners.

For many of the generic container use cases, one could work with Docker Swarm, Kubernetes, and Mesos as all three can be configured to use the same Docker Container Engine [5]–[7]. And since containerization is an operating system feature, all container engines would have to use the same kernel interfaces to initialize and manage containers on a particular operating system [11]. Hence, in this section, we will review some of the different methods orchestration tools use to solve problems, including performance and scalability. Also, we shall be comparing their ease of use, their community support and their popularity with data center operators as well as application developers.

A. Performance and Scalability

Virtual machines will obviously be slower than containers [11]. Therefore we shall not be comparing the performance of virtual machines against containers. However, we shall be comparing the various container orchestration tools: Docker Swarm, Kubernetes, and Mesos [5]–[7].

The developers of Docker Swarm claim that it can scale up to tens of thousands of containers with little performance degradation, whereas Kubernetes is an order of magnitude less scalable [5]. Benchmarks by third-party entities have proven that Docker Swarm provides much better performance than Kubernetes [25]. Kubernetes does have other advantages as described in the following sections, however.

Mesos, like Kubernetes, performs load balancing and autoscaling based on the CPU usage of the machine [7]. While the scalability of Mesos has not been extensively evaluated, at least when compared to Kubernetes, Mesosphere has simulated Mesos to scale to tens of thousands of nodes [26].

Due to a lack of infrastructure, we have not run the benchmarks ourselves. However, a later paper could entail more detailed performance and scalability benchmarks and analysis.

B. Protocols used

In all orchestration tools that we have discussed in this paper, the communication between the manager node and the worker nodes is secured using standard encryption [**?**], [2]–[5], [7]. The Ansible server rightly uses the SSH protocol to communicate with the Ansible clients [4]. Therefore, it is necessary to configure public-private key-based authentication for Ansible to function. The Chef server uses a custom network protocol to communicate with the Chef clients [3]. OpenStack uses the hardware firmware to boot up via the high-speed InfiniBand network that is typically used in data centers [1], [2].

The Docker Swarm manager provides a DNS name for every container in the cluster [16]. Using the discovery service backend and by creating overlay networks, the Docker swarm and the other Docker nodes can easily communicate with one another. In Kubernetes, The group of containers running on a host is called a pod. The Kubernetes network model allows a pod to communicate with any other pod in the cluster.

Like Docker Swarm, Kubernetes also creates an overlay network for this to be feasible [18]. However, Mesos primarily makes use of Network Address Translation to facilitate communication between containers. Since ports on a physical machine are a limited resource, the size of the cluster is severely constrained. However, by integrating a third-party tool into Mesos, an IP address can be assigned to every container, enabling easier communication [27].

Docker Swarm exposes the same Docker API to administrators [5]. Therefore, the same frontends that use the Docker Container Engine can be used to control the Swarm. Using Docker CLI or Docker Compose [28], the entire cluster can be managed.

Kubernetes and Mesos expose their own interfaces which are, of course, incompatible with Docker's own. Therefore any multi-container application that is described using Docker Compose would require significant modification to its configuration if another orchestration tool like Kubernetes or Mesos were to be used [**?**], [7]. However, the developers of these container orchestration tools, containers, and engines, as well as providers of containers as a service, have recently established a consortium to help maintain compatibility and pool resources to improve containers in general. Better collaboration between the orchestration tools would only assist the field of containers.

The master-slave architecture theoretically has a major flaw; there is a single point of failure. The use of the master-slave architecture [**?**], [5], [7], and a tool like Zookeeper [22] to maintain high availability of the master node does make the use of these orchestration tools for cases that encompass clusters across geographical regions infeasible. Since only one manager node is operational at any given time, decentralizing these orchestration tools would require significant changes to their workings. The use of a consensus algorithm [29], [30] would be necessary for such a system design.

All three container orchestration tools restart malfunctioning containers and do not advertise the presence of the container to the client until the application is ready to receive requests. The cluster's load balancer and autoscaling tool, whichever is configured by the administrator, would automatically ensure that traffic is routed away from failed nodes until the restarted application is ready. The master node is always replicated to provide redundancy [**?**], [7] and a tool like Zookeeper [22] would ensure that the switching over to secondary master nodes would happen with little or no impact on the worker nodes. This way, the master-slave architecture's single point of failure is remedied with a quick recovery process for the master node.

C. Ease of use and other advantages

Ansible and Chef are used to manage applications running on virtual machines on physical machines in a data center [3], [4]. Both are quite similar in their use cases. Chef is more full-fledged as it requires the use of the Chef Developer Kit to configure and describe a cluster [3].

Kubernetes, Docker Swarm, and Mesos all can use the same Docker Container Engine [**?**], [5], [7]. While Mesos also has support for its own Mesos Container Engine, the use of Docker Container Engine is almost ubiquitous in the container community [16]. The use of Docker Container Engine enables the use of the Docker repository for pulling commonly used configurations and applications on Docker. Since the Docker community is so vast, the documentation, technical support, and compatibility are, in our opinion, in a good state.

Google engineers developed Kubernetes for their internal use. It was originally used for building and testing internal Google code on their cluster Borg [**?**], [20]. For more than a decade, Google has successfully used Kubernetes to orchestrate the build and testing of machines

in its cluster. Now with the increasing popularity of containers, Kubernetes is now available for anyone to use for their orchestration needs [21]. With Google's stamp of support and a growing community of administrators and developers, Kubernetes has become the most popular container orchestration tool [31].

Kubernetes is more fully featured than Docker Swarm or Mesos [**?**]. For example, Docker Swarm does not have a built-in monitoring agent that can autoscale the containers based on CPU usage or other factors. However, external monitoring tools can be used with Docker Swarm [5]. Autoscaling can be set up using external tools that interface with the Docker API.

Mesos also has support for autoscaling but is available only for Mesosphere Enterprise customers [32]. Therefore, an external third-party tool is necessary for autoscaling in Mesos as well. Also, the popularity of Docker Swarm and Kubernetes is eclipsing that of Mesos, driving up community usage and vendor support [31].

However, Docker Swarm exposes the same Docker APIs to data center operators [5]. Hence, Docker CLI or Docker Compose [28] can be used to design and describe multi-container applications, which are apt for distributed systems. Docker Swarm is, therefore, the easiest to set up for data center operators. And administrators can pick and choose their third party external tools to integrate into their container engine and orchestration tool to provide the extra features that are necessary.

D. Caveats

Some tools like Chef exhibit vendor lock-in by enforcing the use of the Chef Developer Kit. The description of the desired configuration of the cluster and the network is written in a declarative language that cannot be easily converted to the formats used by Ansible.

Some, like OpenStack, follow the standard set by the community and consortium of cloud software providers. But, OpenStack is for quickly setting up bare metal physical machines and quickly building up a data center or a cloud. OpenStack Heat is the component that orchestrates the setup and management of OpenStack [2].

Documentation is still extremely limited for the container orchestration tools. Since the tools were developed only in the past few years, some of the interfaces are not yet stabilized for long-term use. Therefore whatever frontend the data center operator uses to interface with Docker Swarm or Kubernetes or Mesos, it may quickly become incompatible with a newer release of the orchestration tool or the Docker Container Engine. The use of Docker Container Engine for every container orchestration tool may make it easier to transition to another orchestration tool. However, it would also make the application dependent on Docker itself.

V. Future of Orchestration

Automation and ease of management are necessary for greater efficiency in data center operators, cluster administrators, and distributed application developers. While the current orchestration tools have similar architectures, and also provide similar functionality, there is always scope for improvement. Small gains in performance, scalability, or even ease of use improve the efficiency of operating a distributed cluster. Due to the scale of data centers and the cloud, small gains could translate to significant increases in performance and scalability and significant decreases in power consumption and other operating costs.

There is a dearth of scholarly work in this field, especially regarding the evaluation of modern container orchestration tools in terms of performance and scalability. However, this could be because the tools are still under continuous rapid development, trying to outdo one another. And a paper that comprehensively evaluates them could get quickly outdated. This is particularly the case when some of the components involved in the software infrastructure of container orchestration are also the same for different orchestration tools [16].

With the increasing use of geo-replicated clusters for lower latency for clients across the world [33], orchestration tools that can manage clusters that span across the globe would be necessary. However, the master-slave architecture would not be practical due to the significant network latency involved. Therefore, a more decentralized but yet one where the administrator can control and manage the entire cluster as one entity would be necessary. Perhaps by using a consensus protocol like Paxos [29], or Raft [30], one can attempt to decentralize the system by ensuring that orchestration could be done by communicating any one node in the cluster. The use of Zookeeper [22] for ensuring the high availability of the manager node in the swarm would not be necessary as the points of failure would increase if decentralization is done.

The use of virtual machine orchestration tools like Ansible and Chef shall decrease, however, as modern distributed applications are more suitable to be run in containers. OpenStack and its orchestration component, Heat, will continue to be used to set up physical bare metal machines. Each one would have an operating

system, and a container engine could be configured. Container orchestration tools can then be used to describe and manage them. However, the use of virtual machines will not entirely die out, as detailed in the next section.

A. Use of Virtual Machines

Container Engines running on virtual machines are preferable over running container engines natively on physical machines. This is because live migration of virtual machines is easier and more straightforward than a live migration of a natively running operating system [8]. Hence, resource quotas are hit on the physical node, and migration of virtual machines is performed by the hypervisor to balance the load on the physical machines.

Also, as mentioned previously, virtual machines are necessary for running legacy applications that only run on legacy operating systems and legacy hardware that are very difficult or infeasible to obtain. A legacy distributed application would require setting up the physical machines using OpenStack Heat, Ansible, or Chef for virtual machine and application orchestration. Since containers do run the application natively, albeit with kernel-level isolation, they are not suitable for running legacy applications.

However, the use of nested virtualization [34] would decrease as more applications are run on containers rather than directly on virtual machines or natively. Nested virtualization increases the complexity of the setup and would also make orchestration harder to achieve. Data center operators would want to choose simpler methods that provide the same security and isolation guarantees for running their clients' applications.

VI. Conclusion and Future Work

The use of orchestration tools for managing containers and virtual machines running on many physical machines is only increasing. Orchestration tools provide one important function among many others, and that is the automation of running distributed systems. A distributed system consists of many processes, each running in isolation in its own virtual machine or container. It is necessary for the administrator to control and manage the entire distributed system as a single entity.

While the dearth of scholarly work in this field has not hampered the increasing popularity and the development of container orchestration tools, it has made it harder for data center operators, administrators, site reliability and test engineers, and even application developers to make an informed choice about their orchestration tool. There are many whitepapers and articles by the developers of the orchestration tools themselves that explain and evaluate their tool. After reviewing some of the evaluations performed by the developers themselves, they are, in our opinion, subject to bias. Hence, a forthcoming paper that carries out a comprehensive performance and scalability evaluation would be desired.

Benchmarking tools for container orchestration are also not very popular at the time of writing this paper. They are still under heavy development or are poorly documented. A future paper that describes a comprehensive benchmarking tool that can interface with various container engines and container orchestration tools would also be desired.

Orchestration, in general, for distributed systems is not new; many papers have been written on the subject. For example, one describes how a geo-replicated or a cluster distributed across a large geographical distance could be orchestrated using the internet while maintaining security [35]. However, the purpose of this survey was to help data center operators choose the right container orchestration tool.

References

[1] Mellanox Technologies, "Introduction to InfiniBand," 2003.

[2] Openstack, "Adding speed agility to enterprise virtualization: Modernizing virtualized infrastructures with the OpenStack cloud management platform," 2015.

[3] "Chef," https://www.chef.io/chef/, accessed: 2016-12-09.

[4] R. Hat, "Ansible in depth: A whitepaper," 2016.

[5] "Docker Swarm," https://www.docker.com/products/docker-swarm, accessed: 2016-12-09.

[6] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *Queue*, vol. 14, no. 1, pp. 10:70–10:93, Jan. 2016. [Online]. Available: http://doi.acm.org/10.1145/2898442.2898444

[7] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 295–308. [Online]. Available: http://dl.acm.org/citation.cfm?id=1972457.1972488

[8] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, ser. NSDI'05. Berkeley, CA, USA: USENIX Association, 2005, pp. 273–286. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251203.1251223

[9] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 164–177, Oct. 2003. [Online]. Available: http://doi.acm.org/10.1145/1165389.945462

[10] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *SIGACT News*, vol. 33, no. 2, pp. 51–59, Jun. 2002. [Online]. Available: http://doi.acm.org/10.1145/564585.564601

[11] Serverspace, "Introduction to Containerisation," 2015.

[12] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," Feb 2009. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

[13] "Linux cgroups," https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt, accessed: 2016-12-09.

[14] "Hypervisor.framework," https://developer.apple.com/reference/hypervisor, accessed: 2016-12-09.

[15] "Hyper-V Containers," https://msdn.microsoft.com/en-us/virtualization/windowscontainers/management/hyperv_container, accessed: 2016-12-09.

[16] "What is Docker?" https://www.docker.com/what-docker, accessed: 2016-12-09.

[17] "What is Kubernetes?" http://kubernetes.io/docs/whatisk8s/, accessed: 2016-12-09.

[18] "Platform9: Container Orchestration Tools: Compare Kubernetes vs Docker Swarm," https://platform9.com/blog/compare-kubernetes-vs-docker-swarm/, accessed: 2016-12-09.

[19] "Overview of Chef," https://www.chef.io/chef/, accessed: 2016-12-09.

[20] A. Verma, L. Pedrosa, M. R. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France, 2015.

[21] "Kubernetes on GitHub," https://github.com/kubernetes/kubernetes, accessed: 2016-12-09.

[22] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: Wait-free coordination for internet-scale systems," in *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIXATC'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 11–11. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855840.1855851

[23] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[24] A. Williams, B. Ball, H. Dinh, S. Charrington, J. Jackson, J. Williams, L. Hecht, L. Lefler, and M. Maher, "Automation orchestration with Docker containers." The New Stack, 2016. [Online]. Available: https://mesosphere.com/wp-content/uploads/2016/06/TheNewStack_Book3_Automation_and_Orchestration_with_Docker_and_Containers-1-1.pdf

[25] "Evaluating container platforms at scale," https://medium.com/on-docker/evaluating-container-platforms-at-scale-5e7b44d93f2c, accessed: 2016-12-09.

[26] "Visualizing 50000 live containers," https://mesosphere.com/blog/2015/10/06/visualizing-50000-live-containers-how-we-built-our-mesoscon-15-demo/, accessed: 2016-12-09.

[27] "Platform9: Container Orchestration Tools: Compare Mesos vs Docker Swarm," https://platform9.com/blog/compare-kubernetes-vs-mesos/, accessed: 2016-12-09.

[28] "Docker Compose," https://docs.docker.com/compose/, accessed: 2016-12-09.

[29] L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, no. 2, pp. 133–169, May 1998. [Online]. Available: http://doi.acm.org/10.1145/279227.279229

[30] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIX ATC'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 305–320. [Online]. Available: http://dl.acm.org/citation.cfm?id=2643634.2643666

[31] "Open Hub," https://www.openhub.net/, accessed: 2016-12-09.

[32] "Mesosphere," https://mesosphere.com/, accessed: 2016-12-09.

[33] C. Li, D. Porto, A. Clement, J. Gehrke, N. Preguiça, and R. Rodrigues, "Making geo-replicated systems fast as possible, consistent when necessary," in *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. Hollywood, CA: USENIX, 2012, pp. 265–278. [Online]. Available: https://www.usenix.org/conference/osdi12/technical-sessions/presentation/li

[34] M. Ben-Yehuda, M. D. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B.-A. Yassour, "The turtles project: Design and implementation of nested virtualization," in *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*. Vancouver, BC: USENIX Association, Oct. 2010. [Online]. Available: https://www.usenix.org/conference/osdi10/turtles-project-design-and-implementation-nested-virtualization

[35] J. Misra and W. R. Cook, "Computation orchestration," *Software & Systems Modeling*, vol. 6, no. 1, pp. 83–110, 2007. [Online]. Available: http://dx.doi.org/10.1007/s10270-006-0012-1