A Survey on Virtual Machine and Container Orchestration

Aneesh Neelam
Univ. of California, Santa Cruz
aneelam@ucsc.edu

December 8, 2016

Abstract

Data center administrators and site-reliability engineers create virtual machines or containers to run their applications, with desired redundancy requirements and automated coordination between the replicas. This is called the orchestration of the various computing, storage and network resources of the data center. To automate the orchestration process, different tools have been developed in the past couple of decades. Each tool was designed differently, with different intents and priorities but they also have a lot in common. This survey is on the various orchestration tools available to data center administrators, site reliability engineers and software engineers; and aims to be a comprehensive guide that helps data center administrators in choosing the cloud platform and the orchestration tool.

1 Introduction

With the advent and proliferation of virtualization, computing as a service has taken off. Now there are multiple cloud service providers such as Google Cloud, Amazon Web Service, Microsoft Azure, etc. 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) [14]. 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 [15]. There are many other such tools such as Chef and Ansible for virtual machines [1,12]. Container Orchestration tools such as Docker Swarm, Kubernetes, Mesos have been developed to manage containers on mostly Linux machines [2,9,13]. In this survey however, we shall focus more on orchestration of containers than virtual machines.

Using these orchestration tools, data center operators can boot up virtual machines or containers on the physical machines, and control the entire the data center as a single entity, provision the necessary resources, and deploy and managing applications. These orchestration tools can also be used to automate 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 [8,10]. Some orchestration tools also enforce quotas for processes, ensure that Service-level agreements are adhered to and can also be used to keep track of the overall resource utilization and power consumption of the data center.

In this survey, the performance of these orchestration tools shall be compared. Also, these orchestration tools may also have different guarantees of tolerance when there is a failure or network partition. According to the CAP theorem, only two out of three: presence of network partitions, availability and consistency can be guaranteed by any distributed system [11]. In which case, there may be different methods the orchestration tools employ for error recovery and maintain high reliability, offering various combinations of two out of the three things for applications to 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 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. We then discuss possible future work in the field in Section 5, and we then conclude in Section 6.

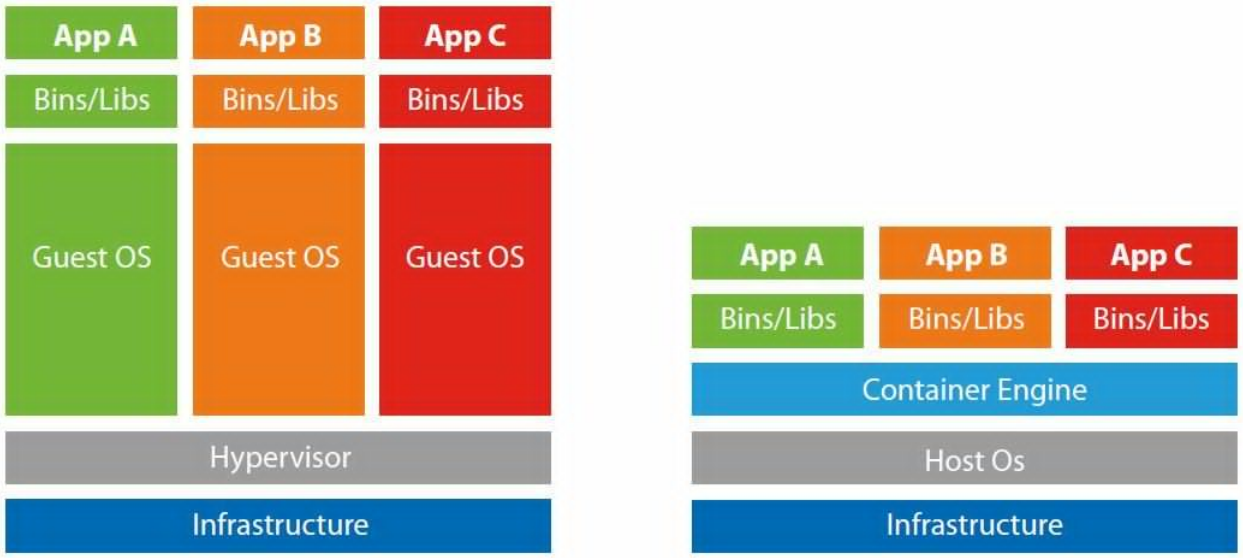

Figure 1: Differences between Virtual Machines and Containers [16]

2 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 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 [7]. Virtualization, in this context, is the task of creating virtual hardware and running actual operating systems and applications on top of them. Each of these virtual machines is isolated and independent from one another and are managed by a hypervisor that is running on the physical hardware or on the host operating system [8].

Virtualization technology is now decades old, with every major operating system that runs on servers having 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 is going on with containers on these operating systems as well. Like Cgroups in the Linux kernel, Hypervisor.framework in Apple macOS and Hyper-V in Microsoft Windows. Container enngines 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 [5].

Virtual machines are a complete hardware virtualization, and there is a full-fledged guest operating system on which applications are run. Virtual machines 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 ioncrese of performance can be achieved with some changes to the guest operating system, but it still does not entail modifying the application [8].

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 [16]. However, that would also mean that the applications would have 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 [16]. 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 [15], Chef [1] and Ansible [12]. Some of the tools for containers we shall cover in this survey are Docker Swarm [2], Kubernetes [9], Mesos [13].

3 General Comparison

As we have seen in the previous section, virtual machines are full-fledged operating systems, each that contain their own set of libraries and run 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 [15] or Ansible [12] or Chef [1] can be used to manage all the hypervisors of a cluster of physical machines, thereby controlling the entire data center as one entity.

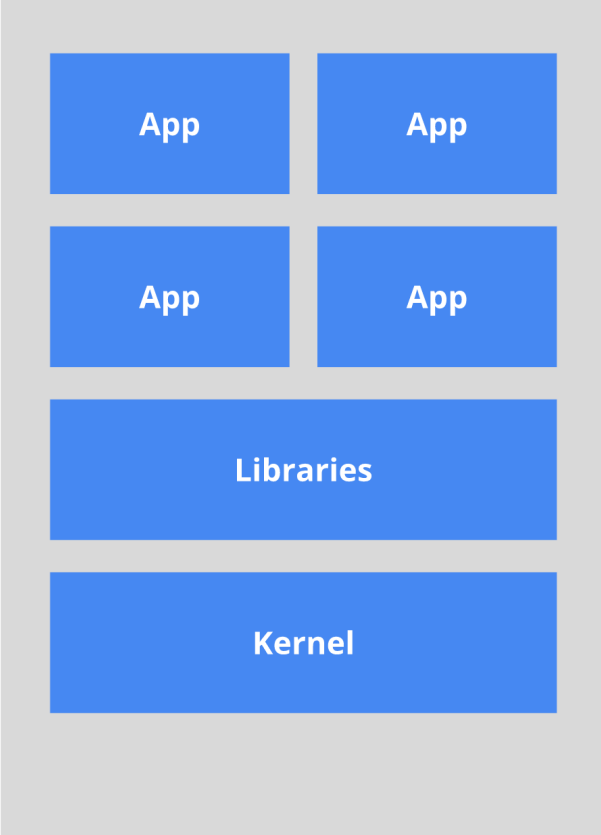

Figure 2: Applications running natively on OS with shared libraries [6]

However, containers have one key advantage greatly increased their popularity and that is their significantly lower overhead when 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 is of a single version as depicted in Figure 2 [6]. One can run the applications in different contexts by running each one in its own virtual machine, but the overhead of virtualization is very high. Containers allow the kernel-level isolation of applications **and their libraries**. Figure 3 illustrates the use of containers for this use case [6].

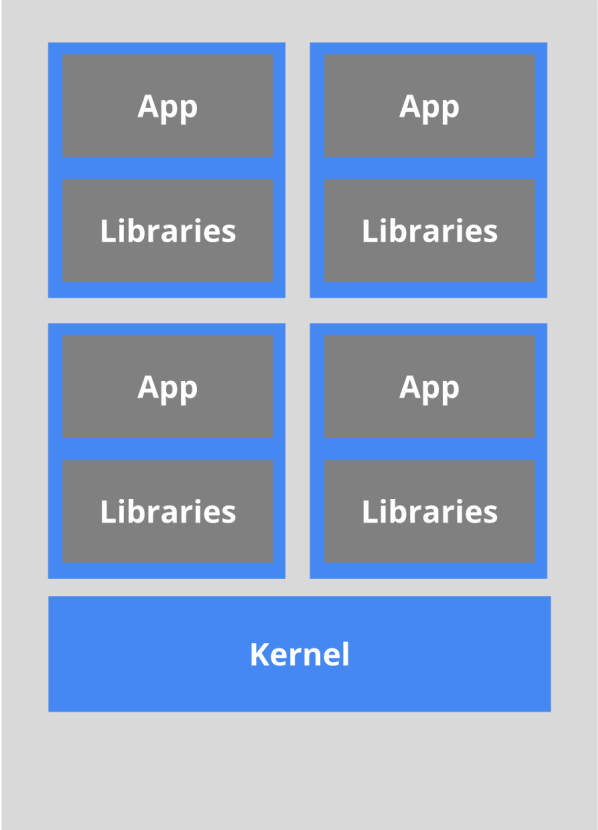

Figure 3: Containers of applications and libraries running on OS [6]

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.

3.1 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 especially true when running legacy applications that require legacy operating systems and legacy hardware. But 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 [12]. It is made up of many components, not all relevant for this survey. We shall focus on the orchestration capabilities of Ansible. An admininstrator can execute hundreds or thousands of virtual machines, specifying an operating system and application image with a complex network configuration set up between them. Configuration is predetermined by the adminstrator, written into config file(s) and specified when running the orchestration tool.

3.2 Container Orchestration

We shall compare Docker Swarm [2], Kubernetes [9] and Apache Mesos [13]. Docker Swarm, Kubernetes and Mesos all have a master-slave architecture, where a small number of manager nodes control the rest of the cluster [2, 9, 13].

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 nodes of the cluster, 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 case of a failure. [2]. Third party tools that maintain high availability of the swarm manager can be integrated into this setup [4].

One of the key advantages of Docker Swarm is how 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 [4].

Kubernetes is a orchestration tool developed by engineers at Google. Like Docker Swarm, Kubernetes also works with the Docker Container Engine [9]. It was originally for internal use only and was designed to orchestrate the use of their internal cluster, Borg [17]. 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 [3]. As illustrated in Figure 5, Kubernetes has a master-slave architecture [4]. Pods are groups of containers that run on the nodes in the cluster [9].

4 Evaluations

4.1 Performance

4.2 Ease of Use

4.3 Protocols used

4.4 Use Cases

4.5 Caveats

5 Future of Orchestration

5.1 Use of Virtual Machines

5.2 Further Innovation and Development

6 Conclusion

References

[1] Chef. https://www.chef.io/chef/. Accessed: 2016-12-09.

[2] Docker Swarm. https://www.docker.com/products/docker-swarm. Accessed: 2016-12-09.

[3] Kubernetes on GitHub. https://github.com/kubernetes/kubernetes. Accessed: 2016-12-09.

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

[5] What is Docker? https://www.docker.com/what-docker. Accessed: 2016-12-09.

[6] What is Kubernetes? http://kubernetes.io/docs/whatisk8s/. Accessed: 2016-12-09.

[7] 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.

[8] 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.*, 37(5):164–177, Oct. 2003.

[9] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes. Borg, omega, and kubernetes. *Queue*, 14(1):10:70–10:93, Jan. 2016.

[10] 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*, NSDI'05, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.

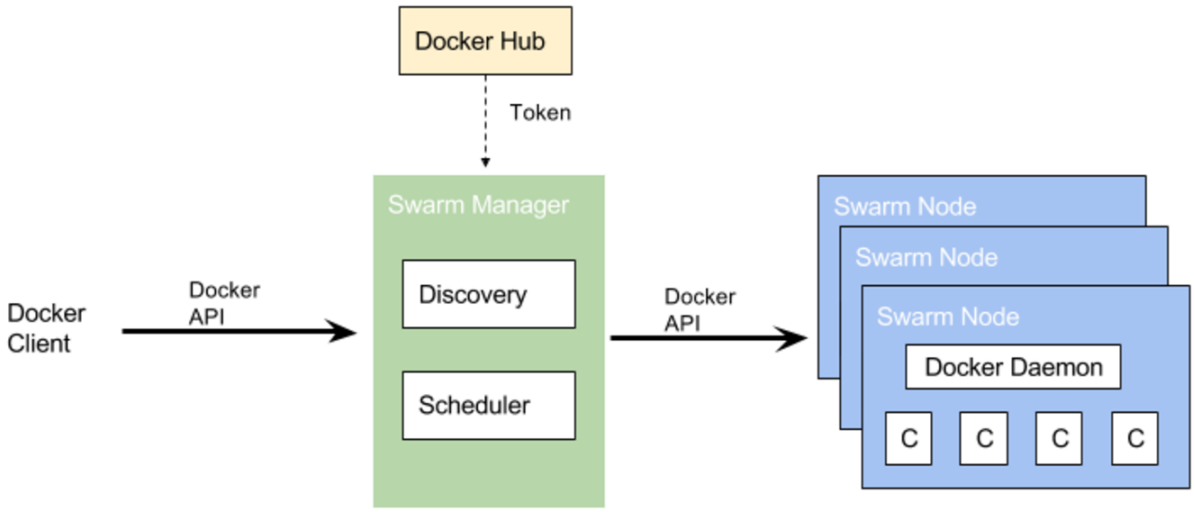

Figure 4: Overview of Docker Swarm [4]

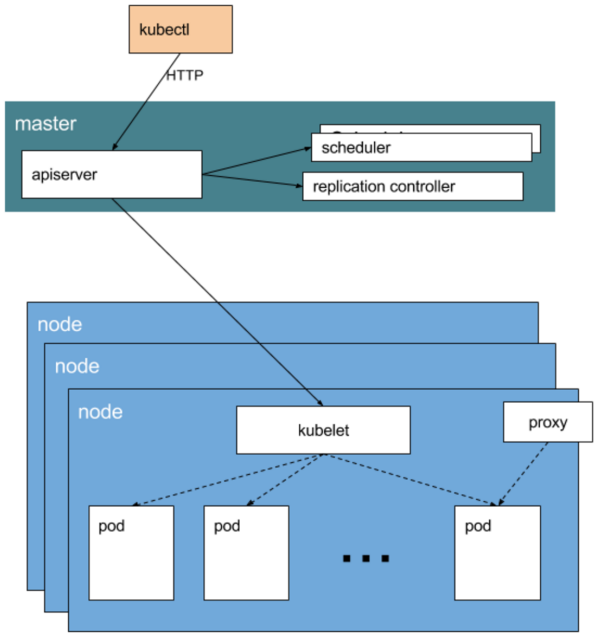

Figure 5: Overview of Kubernetes [4]

[11] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, June 2002.

[12] R. Hat. Ansible in depth: A whitepaper. 2016.

[13] 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*, NSDI'11, pages 295–308, Berkeley, CA, USA, 2011. USENIX Association.

[14] Mellanox Technologies. Introduction to InfiniBand. 2003.

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

[16] Serverspace. Introduction to Containerisation. 2015.

[17] 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.