International Journal of Core Engineering & Management
Volume-6, Issue-06, 2019 ISSN No: 2348-9510

HARNESSING CONTAINER ORCHESTRATION WITH DOCKER AND KUBERNETES FOR APPLICATION DEPLOYMENT TO BUILD A SCALABLE APPLICATION

Satyadeepak Bollineni
DevOps Engineer
Databricks
Texas, USA
Email: deepu2020@gmail.com

Abstract

Container orchestration has revolutionized how scalable applications can be deployed and managed, especially in cloud environments. This work discusses how two open-source giants in container orchestration, namely Docker and Kubernetes, can be merged to enhance application deployment scalability. The paper further delves into best practices in deploying scalable applications, covering the basics of containerization, Kubernetes architecture, and the interaction between Docker and Kubernetes. Case studies are used to illustrate how these technologies have been applied in practice. Moreover, the paper emphasizes the importance of future directions in container orchestration, inspiring the audience to think forward and innovate in this field.

Keywords: *Container orchestration, Docker, Kubernetes, scalable applications, cloud computing, deployment.*

I. INTRODUCTION

The advent of Microservices and containerization has brought about a significant transformation in the software development industry. The pressing need for scalable, resilient, and efficient application deployment has been the driving force behind this transformation. Containers, which encapsulate applications along with their dependencies, have emerged as a key element in ensuring consistency across diverse environments. In this context, Docker, introduced in 2013, has become a leading platform for the automation and deployment of these containers. As applications continue to grow in complexity, the demand for tools that facilitate their effective management has become more pronounced, leading to the development of container orchestration platforms like Kubernetes.

Large-scale container management presents several challenges: deployment, scaling, load balancing, and fault tolerance. Without a robust orchestration tool, these tasks can be cumbersome, inefficient, and sometimes lead to downtime. The integration of Docker with Kubernetes offers real power by automating many operational tasks essential for efficiently managing containerized applications. This paper not only discusses the theoretical aspects but also demonstrates how Docker and Kubernetes enable deployable scalability, providing a deep understanding of how such tools can be practically used to build reliable and scalable applications.

International Journal of Core Engineering & Management
Volume-6, Issue-06, 2019 ISSN No: 2348-9510

II. BASICS OF CONTAINERIZATION

1. Concept of Containers

The concept of containers, rooted in the early Unix era with chroot environments, provided the first foundational work on isolating file systems for specific processes. Further developing this idea was the introduction of Linux Containers, or LXC, in 2008, which began modern containerization [1]. Containers are lightweight, independent, and executable software packages that include everything needed to run the application, which embodies code, runtime, system tools, libraries, and settings. This provides self-contained consistency and reliability across environments from development into production. Unlike the VMs, which would take one whole OS for every VM, containers share the host operating system's kernel. Because of this, containers have become very lean regarding resources due to shared-kernel architecture and thus fire up quickly. This model allows more containers to run over the same physical or virtual machine, essential in environments with limited resources.

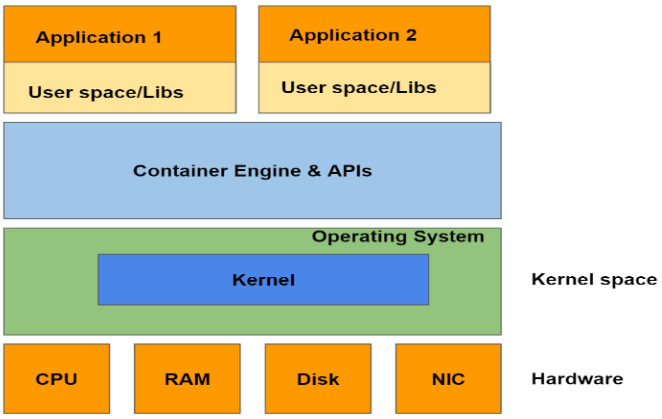


Figure 1: Container Architecture [1]

The above figure provides a generalized view of container architecture: Containers package an application and its dependencies, including libraries, and then run the deployed applications in user space. The container engine and APIs interact with the kernel of the operating system underneath, which controls the system hardware- the CPU, RAM, hard disk drive or solid-state drive, NIC, and so forth. Such a structure offers an efficient, predictable, and isolated environment for running applications.

More importantly, constituent parts of these containers are light and have portability and scalability implications, as well as features glimpsed in today's fast-evolving developments. Therefore, a containerized application can seamlessly move to testing, staging, and production from a developer's local environment with no cause for worry concerning system compatibility or environmental differences. [2]. This portability is one modern cornerstone for DevOps practices, emphasizing continuous integration and deployment. This notion may enable a microservice to run independently in each container, making it easy to scale, upgrade, and manage each service. This is fundamentally useful and far-reaching, as modular design dramatically improves the agility and flexibility of applications; the failure of one single container does not cascade to the rest of the containers, thereby ensuring system stability and reliability. The features of being portable, efficient, scalable, and isolated have made containers an inseparable part of modern software development and deployment, hence their extensive industrial consideration for employment.

International Journal of Core Engineering & Management
Volume-6, Issue-06, 2019 ISSN No: 2348-9510

2. Docker

The Docker Container Platform was one of the very first containerization platforms. In 2013, it made the management and distribution of containers pretty simple for developers. Docker's three broad components are the Docker Engine, Docker Hub, and Docker Compose. The Docker Engine is responsible for running and building containers, providing it with all the utilities and tools necessary to give an application a complete runtime environment. [3]. Docker Hub can act like a cloud-based registry where people can share and download container images, enhancing collaboration or reuse of containerized applications.

Figure 2 : Layered Architecture of Docker[2]

The above image describes the layered architecture of Docker, drafted on the need for core management. The Server Docker Daemon forms the base for managing containers, networks, images, and data volumes. The Rest API and Client Docker CLI interact with the user and daemon for smooth communication and hence manage and deploy containers through a managed interface. [3].

Besides, Docker Compose is responsible for defining and running multi-container Docker applications, which means complex applications comprising multiple interconnected services are easy to handle. Such tools made deployment easy because a developer can now package an application with all its dependencies into a single portable image that can later be deployed consistently across diverse environments. Then, the application behaves precisely identically on your laptop as on a production server. Docker's focus on portability, consistency, and isolation had it set as the de facto standard for containerization. This set up an ecosystem around quick and efficient ways to push applications out. Consequently, Docker has become an integral part of the DevOps toolkit, which enables continuous integration and implements continuous deployment practices to allow the speed and reliability that drives organizations in software delivery.

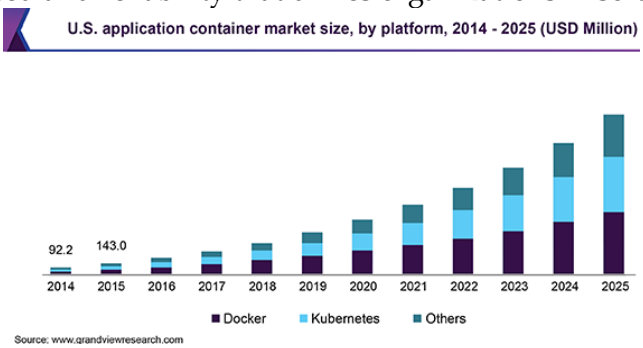


Figure 3 : US Application Container [3]

International Journal of Core Engineering & Management
Volume-6, Issue-06, 2019 ISSN No: 2348-9510

The image above shows the USD million size of the U.S. application container market by platform from 2014 to 2025. This chart reflects the growth of Docker, Kubernetes, and other platforms over the years; the itinerary has ousted both Docker and Kubernetes as leading the market. The projected market size will surge considerably by 2019, while Kubernetes will continue to increase by 2025.

III. INTRODUCTION TO CONTAINER ORCHESTRATION

1. Need to Orchestrate

With an increase in the adoption rate in organizations, the scale at which containers are deployed has increased to such a level that their management, manually, has become highly limited. Manual management of hundreds, sometimes thousands of containers became impractical and error-prone; such inefficiencies increased the risk of downtime. This orchestration of containers encompasses the deployment of containers, health checking, scaling up or down depending on demand, and automatic recovery in case of failures. If these actions were to be done without orchestration, they would need human intervention, including many delays and disturbances in service. Besides, modern applications are highly dynamic, and rapid updates are often required to face fluctuating loads. This contemporary need for an effective orchestration solution made it all the more critical. Out of this need came a class of tools called container orchestration, which aimed at automating these processes and ensuring that containerized applications could be managed efficiently, securely, and reliably at scale [4].

2. Overview of Orchestration Tools

Several container orchestration tools have emerged over the years, spanning a spectrum from strong to weak. First orchestration platforms: Docker Swarm is a native clustering and scheduling tool for Docker containers. Due to an outstanding level of flexibility and scalability and very active community support since the open-source announcement by Google in 2014, Kubernetes attained a position among the leading orchestration tools very swiftly. Kubernetes provides a compelling platform that automates application container deployment, scaling, and operation across cluster hosts. It abstracts the underlying infrastructure so developers focus on building applications, not managing the underlying systems.

IV. KUBERNETES: LEADING ORCHESTRATION PLATFORM

1. Overview of Kubernetes

Kubernetes- abbreviated as K8s- is an open-source platform designed to manage containerized applications that run on a set of hosts. It provides a standard API for deploying, scaling, and operating applications. Kubernetes builds upon over 15 years of experience running containerized applications at Google, combined with the best-of-breed ideas from the community. The fundamental building blocks of Kubernetes are Pods, Nodes, Clusters, and Services. A Pod is the smallest deployable unit in Kubernetes, comprising one or more containers that share the same network namespace. Each Pod runs on a Node, which is a physical or virtual machine. A Cluster is a set of Nodes managed by the Kubernetes control plane. [5]. A service is an abstraction that defines a logical set of pods and a policy to access them, thereby enabling load balancing and service discovery.

International Journal of Core Engineering & Management
Volume-6, Issue-06, 2019 ISSN No: 2348-9510

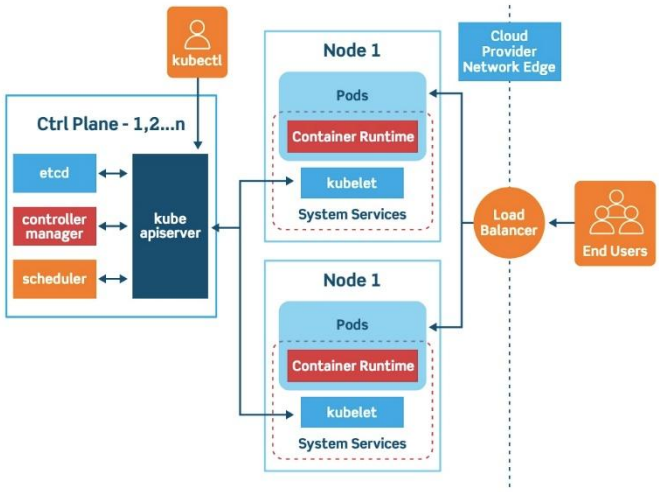


Figure 4 : Kubernetes Architecture [5]

The above diagram illustrates the high-level view of conceptual Kubernetes architecture. Kubernetes controls its containers by using a central API server that is responsible for tracking the current state of the Kubernetes objects. Controllers and Schedulers determine what needs to happen and execute actions. In contrast, Kubelets manage container runtimes on worker nodes, ensuring that the running containers are as specified in the desired state defined by Kubernetes. [5].

2. Important Features of Kubernetes

Kubernetes has several critical features that make it such a powerful orchestration platform. Key among them is its core strength, scalability, which enables applications to scale up or down automatically due to demand. Kubernetes deals inherently with load balancing in a way that distributes traffic evenly across Pods. Self-healing by nature, Kubernetes can replace failed pods automatically or reschedule them for high availability on healthy nodes. With multi-cloud and hybrid cloud in action, Kubernetes supports both and stands tall for an organization seeking to deploy applications across infrastructures provided by different vendors. Kubernetes abstracts the underlying infrastructure to consistently deploy across on-premise data centers, public clouds, and edge environments.

3. Kubernetes Architecture

Kubernetes architecture is based on the master-worker model. The control plane components include the API Server, Scheduler, and Controller Manager. The API Server is the front end of the Kubernetes control plane, which generally completes the requests from users and components. The Scheduler decides which Node other new Pods should run, while the Controller Manager manages the overall state of the cluster and checks that the desired state matches the current state. Worker Nodes, sometimes called Minions, execute the application workloads. Each node will contain one Kubelet responsible for communicating with the API server and ensuring that the containers in the pods are running as intended. The Kube-proxy component enables network connectivity and load balancing for Pods [6].

International Journal of Core Engineering & Management
Volume-6, Issue-06, 2019 ISSN No: 2348-9510

V. INTEGRATION OF DOCKER AND KUBERNETES

1. How Docker Works with Kubernetes

These two technologies, Docker and Kubernetes, complement each other. Docker provides the runtime for the containers, while Kubernetes orchestrates them. Docker does the low-level container runtime for Kubernetes. It leverages Docker's capability to build, package, and run containers. Therefore, developers can keep working with the tooling and workflows they're used to from Docker but still get value from the powerful orchestration capabilities of Kubernetes. It uses Docker to deploy an application with the creation and management of containers running the application code. Kubernetes then orchestrates these, ensuring they are deployed, scaled, and managed according to the application requirements. Such integration makes deployment more accessible and frees the developers to build applications rather than manage infrastructure. [7].

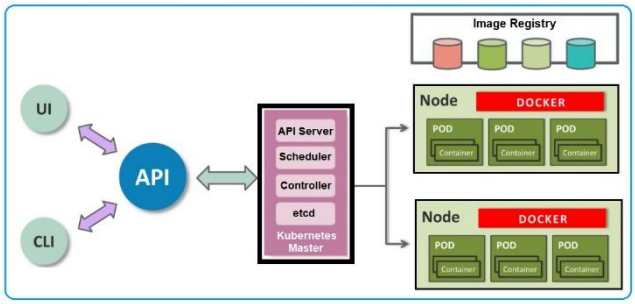


Figure 5 : Docker and Kubernetes Integration [3]

The above diagram shows their integration: Docker and Kubernetes Integration. Kubernetes orchestrates containerized applications by communicating through its master node to the Docker containers comprising the API Server, Scheduler, and Controller. Docker handles the containers within Pods across various nodes; Kubernetes deploys, scales, and manages them. The container images for deployment are stored in the image registry. [8].

2. Using Docker with Kubernetes: Benefits

The integration of Docker with Kubernetes has the following advantages in application deployment. This will enable faster and more consistent deployment. The lightweight containers from Docker can be deployed more quickly into diverse environments. Besides, Kubernetes can automate scaling high-level abstractions in a similar way to how maintenance of objects is performed. This is simply because of two significant factors: combining Docker with Kubernetes allows for greater scalability. Kubernetes scales applications automatically on demand, whereby resources are efficiently used. Such scalability is of utmost importance in applications whose accesses vary in traffic; typical examples include e-commerce platforms and video streaming services. Third, Docker and Kubernetes increase an application's reliability and high availability. Because of the self-healing process, the containers' failure is automatically replaced by Kubernetes. Hence, this reduces the overall downtime of an application and increases its resilience. [9].

VI. CASE STUDIES: SCALABLE APPLICATIONS USING DOCKER AND KUBERNETES

1. Case Study 1: E-commerce Platform

One typical project using Docker and Kubernetes can be imagined as an e-commerce platform that experiences significant traffic slashes seasonally, particularly during peak shopping seasons. Such a platform would follow a microservices architecture, with each service running in its container.

International Journal of Core Engineering & Management
Volume-6, Issue-06, 2019 ISSN No: 2348-9510

Kubernetes orchestrates the actual deployment of these containers across a cluster of nodes. For example, when traffic loads hit the application during Black Friday, Kubernetes automatically scales the services by dynamically scaling them up. [10]. This means this platform will remain responsive and available to customers during such times. Another good thing is that using Docker and Kubernetes made deployment easy so that we can roll out new features quickly and reliably.

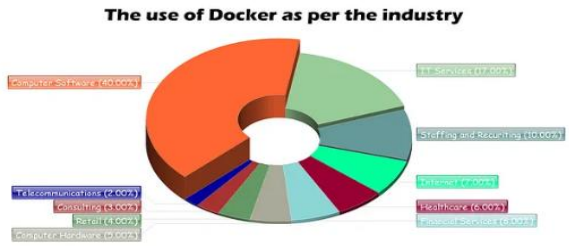
Figure 6 : Percentage of organizations [6]

The chart above from the same survey shows the percentage of organizations using different container management tools as of Fall 2017. As one might expect, given its popularity in the marketplace, Kubernetes has a significant lead, managing containers for 69% of surveyed organizations. Next in line was Amazon ECS at 20% and Docker Swarm at 18%. Google Container Engine, GKE reached 17%, and of those GKE users, 85% used Kubernetes in generic form. The remainder of the tools depicted are Azure Container Service, Openshift, and shell scripts; these are represented yet show a lower degree of adoption. [11].

2. Case Study 2: Financial Services Application

Another good example could be the financial services application that requires high availability and, at the same time, low latency. The application will be deployed on Docker and Kubernetes, while Kubernetes manages the orchestration of the containers within a geographically distributed cluster. Kubernetes support for a multi-cloud environment enables the application to deploy multiple data centers seamlessly for high availability and disaster recovery. This is particularly useful for this application because Kubernetes allows it to balance loads effectively and provide self-healing in case of any hardware failure or network disruption. Comprehensive integration with Docker also offers rapid deployment of updates and patches, reducing the risk of security vulnerabilities. [11].

International Journal of Core Engineering & Management
Volume-6, Issue-06, 2019 ISSN No: 2348-9510



Computer software covers the maximum, i.e. 40%, followed by IT services, i.e. 17%. While Staffing and Recruiting industry is 10%. This is followed by Internet (7%), Healthcare (6%), Financial Services (6%), Computer Hardware (5%), Retail (4%), Consulting (3%), and Telecommunications (2%).

Figure 7 : The use of Docker as per the industry [10]

VII. CHALLENGES AND CONSIDERATIONS

1. Problems with Container Orchestration

Along with the many benefits that container orchestration brings, it also faces several challenges. The shared kernel of a host with containers has been brought up as a massive security vulnerability that can expose the containers to some attack. Containerized applications must be secured by using best practices for security, offering sound image access controls, and updating containers promptly. Networking is another challenge in container orchestration. Considering a cluster comprises multiple containers, they must communicate among themselves and with other systems outside the cluster.[11]. Kubernetes has several networking solutions, but network configurations and security hardening remain challenges, especially across large-scale deployments. Another vital consideration about container orchestration is persistent storage. By default, containers are meant to be temporary; thus, no data contained within are expected to outlive them, which will call for persistent storage solutions where volumes hold the application data under the destructions of the container. Kubernetes supports persistent volumes, but managing these volumes can be difficult, especially in distributed environments.

2. Best Practices for Effective Orchestration

A few of the best practices for orchestration using Docker and Kubernetes will be presented when addressing such challenges. This includes using only secure, up-to-date container images. The Docker Hub has a massive library of prebuilt photos, but they should undergo an appraisal for security vulnerabilities. Moreover, it will secure the cluster by allowing access only to specific resources with RBAC (Role-Based Access Control) in Kubernetes that can be implemented. RBAC will enable administrators to define fine-grained permission, where access is ensured based only on what the users and services need. This looks into the need for monitoring and logging to maintain a healthy and performing well Kubernetes cluster. Prometheus and tools like Grafana can help monitor the metrics of the entire cluster, while centralized logging solutions are in place to keep track of the activities of the cluster and aid troubleshooting by an administrator.

VIII. THE FUTURE AND BEYOND WITH CONTAINER ORCHESTRATION

1. Evolving Landscape

The container orchestration landscape can only be viewed as dynamically reshaped with new tools and technologies emerging to address the challenges of administering containerized apps. For

example, serverless computing is emerging as an alternative topology to classical container orchestration—serverless abstracts the underlying infrastructure, so developers focus on just code. While Kubernetes and Docker are well into usage, serverless might provide a slightly more efficient and less costly solution for some use cases.

Another approaching trend is the utilization of artificial intelligence and machine learning to optimize container orchestration. Algorithms based on AI and ML inspect the performance of containerized applications, after which they apply resource adjustments to maximize efficiency automatically. This automates the containerized application management process with less—hopefully minimal—manual intervention.

2. Implications on the Future Development of Applications

These changes in container-level orchestration will likely bring one big tidal wave across the fleet of application development. As the level of orchestration goes higher, developers can build increasingly complex and more scalable apps from the same amount of effort put into the development process. Abstracting the infrastructure via orchestration tools enables developers to keep their focus on writing code rather than managing resources. Moreover, orchestration tools with the cloud-native platform would allow deployments to flow smoothly across multi-cloud and hybrid-cloud environments. This means excellent advancement for organizations looking to seize upon best-in-class features from various cloud providers.

IX. CONCLUSION

This paper discussed the role of Docker and Kubernetes in container orchestration, emphasizing the benefits of using them to deploy applications at scale.

Docker provides a lightweight and relatively cohesive container runtime, while Kubernetes automates the orchestration of these containers to make them scalable, reliable, and efficient

REFERENCES

1. Zhang, Q., Liu, L., Pu, C., Dou, Q., Wu, L., & Zhou, W., "A Comparative Study of Containers and Virtual Machines in Big Data Environment," IEEE Xplore, no. https://doi.org/10.1109/CLOUD.2018.00030. pp. 54-65, 2018.
2. Shahin, M., Ali Babar, M., & Zhu, L., "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices.," IEEE Access, vol. 5, no. https://doi.org/10.1109/access.2017.2685629, pp. 3909–3943, 2017.
3. Xenonstack, "Docker Overview, Architecture and Application Deployment.," Medium: Digital Transformation and Platform Engineering Insights, pp. 4-6, July 2017.
4. Tornow, D., "The Mechanics of Kubernetes - Dominik Tornow - Medium.," Medium, pp. 11-15, Nov 2018.
5. Awanish., "Understanding the Kubernetes Architecture with a Use-Case," Medium Edureka., pp. 22-24. Sept 2018.
6. Coleman, M. , "Docker Compose and Kubernetes with Docker for Desktop | Docker," Docker, pp. 12-24. 2018.
7. D. Rabi, "Docker Containers and Kubernetes: An Architectural Perspective. Dzone.com," DZone, pp. 29-35. Nov 2018.
8. Stackify , "The Advantages of Using Kubernetes and Docker Together," Stackify, 19 March 2018.

9. D. Inc., "Autoscaling in docker swarm," Docker Community Forums, pp. 25-30 April 2018.
10. L. U. (. O. 1. I. f. —. C. a. t. you. o. D. M. 4. E. L. U. Education, "Interesting facts — Companies and the use of Docker," Medium, pp. 16-30. Oct 2018.
11. Silva, V. G. da, Kirikova, M., & Alksnis, G., "Containers for Virtualization: An Overview," Applied Computer Systems, vol. 23, no. https://doi.org/10.2478/acss-2018-0003, pp. 21-27, 2018.