International Journal of Computer Trends and Technology
ISSN: 2231–2803 / https://doi.org/10.14445/22312803/IJCTT-V71I8P109

Volume 71 Issue 8, 57-62, August 2023
© 2023 Seventh Sense Research Group®

Original Article

Kubernetes and Docker: An Introduction to Container Orchestration and Management

Sumit Sachdeva

The Scotts Company LLC, Marysville, OH.

Received: 19 June 2023 Revised: 29 July 2023 Accepted: 14 August 2023 Published: 30 August 2023

Abstract *- This paper aims to provide an overview of Kubernetes and Docker, two essential technologies that have revolutionized containerization, orchestration, and management in the world of software development and deployment. We will explore the core concepts, architecture, and functionalities of both Kubernetes and Docker, highlighting their benefits and use cases. Furthermore, we will discuss how these technologies work together to enable scalable, resilient, and portable application deployment on cloud platforms like AWS, GCP, Hana or Azure.*

Keywords *- Cloud infrastructure, DevOps, Business Intelligence (BI), Microservices, AWS, SAP Hana, Azure, CI/CD.*

1. Introduction

The landscape of software development and deployment has undergone a significant transformation in recent years. Traditional monolithic applications have given way to modern, distributed architectures that leverage microservices and containerization. This shift has been driven by the need for greater scalability, flexibility, and efficiency in deploying applications across diverse environments.

Containerization, which enables packaging applications and their dependencies into lightweight and portable units, has emerged as a key technology in this paradigm shift. Docker, a popular containerization platform, has played a pivotal role in simplifying the process of creating, distributing, and running containers. By providing a standardized format for packaging applications, Docker has made it easier to achieve consistency in development, testing, and production environments.

However, as the number of containers in a system grows, managing and orchestrating them becomes increasingly complex. This is where Kubernetes, an open-source container orchestration platform, comes into play. Kubernetes provides a robust set of tools and functionalities for automating the deployment, scaling, and management of containerized applications. It tackles challenges such as container scheduling, load balancing, service discovery, and high availability, making it possible to run and scale applications across clusters of machines efficiently.

2. Docker

Docker is an open-source platform that facilitates containerization, allowing developers to build, package, and distribute applications as lightweight and portable containers. It provides a consistent and reproducible environment for running applications across different systems and environments.

3. Docker Architecture

Docker architecture consists of several components that work together to enable containerization and manage containerized applications. The below figure depicts the architecture of Docker. The main components of the Docker are:

3.1. Docker Client

The Docker Client is the primary interface through which users interact with Docker. It allows users to issue commands to the Docker daemon and manage Docker resources. The Docker Client can run on the same host as the Docker daemon or a remote machine, communicating with the daemon over a REST API.

3.2. Docker Daemon

The Docker Daemon, also known as the Docker Engine, is responsible for managing the Docker objects such as images, containers, networks, and volumes. It listens for Docker API requests from the Docker Client and handles tasks like building and running containers, managing networks, and storing and retrieving images. The Docker Daemon runs in the background on the host machine.

3.3. Docker Images

Docker Images are the building blocks of containers. They are read-only templates that contain the application code, dependencies, libraries, and configuration required to run an application. Images are created using Dockerfiles, which specify the steps to build the image. Docker images are stored in a registry and can be pulled to create containers.

This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)

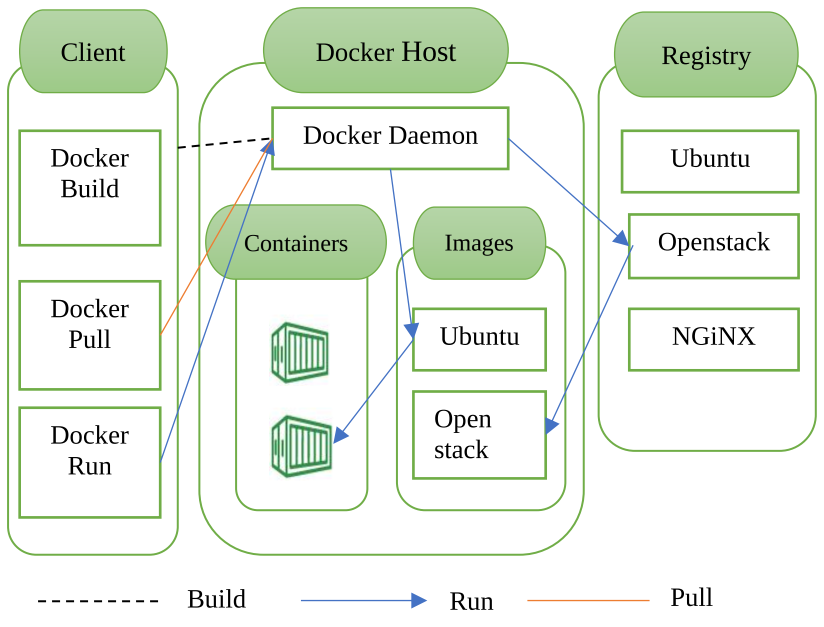

Fig. 1 Docker architecture

3.4. Docker Containers

Docker Containers are instances of Docker Images. Containers are lightweight and isolated environments that encapsulate an application and its dependencies. Each container runs as a separate process with its filesystem, network, and process namespace. Containers can be started, stopped, and restarted independently of each other.

3.5. Docker Registries

Docker Registries are repositories that store Docker Images. They can be public or private. The Docker Hub is the default public registry provided by Docker, where users can find and download pre-built images. Private registries can be set up to store and distribute custom or proprietary images within an organization. Docker images can be pushed to and pulled from registries.

3.6. Docker Networking

Docker provides networking capabilities that allow containers to communicate with each other and the outside world. Each container can be assigned a unique IP address. Docker supports various networking modes, such as bridge networks, overlay networks for multi-host communication, and host networks for direct access to the host's network stack. Docker also supports creating user-defined networks to isolate and manage container communication.

3.7. Docker Volumes

Docker Volumes provide persistent storage for containers. Volumes are directories or files that exist outside the container's filesystem but can be accessed by the container. Volumes allow data to be shared and persisted across container restarts or when containers are replaced. Docker supports different types of volumes, including host-mounted volumes, named volumes, and anonymous volumes.

3.8. Docker Compose

Docker Compose is a tool that defines and manages multi-container applications. It uses a YAML-based file format to describe the services, networks, and volumes needed by an application. Docker Compose simplifies the orchestration of multiple containers and provides a way to define and manage the entire application stack.

4. Kubernetes

Kubernetes is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. It provides a robust and scalable framework for managing containers across a cluster of nodes, offering features such as automated deployments, scalability, self-healing, service discovery, and configuration management. With its powerful architecture and extensive ecosystem, Kubernetes has become the de facto standard for container orchestration, enabling organizations to run and manage modern applications at scale efficiently.

5. Kubernetes Architecture

The architecture of Kubernetes is designed to provide a scalable and resilient platform for managing containerized applications. It consists of several key components that work together to enable the orchestration and management of containers across a cluster of nodes. Figure (see Figure 2) depicts the architecture of Kubernetes. The main components of the Kubernetes are:

5.1. Master Node

The Master Node is the control plane of the Kubernetes cluster. It manages the overall cluster state and orchestrates containerized applications. The key components of the Master Node include:

5.1.1. API Server

Acts as the central management point for the Kubernetes cluster. It exposes the Kubernetes API, which allows clients to interact with the cluster.

5.1.2. Scheduler

Assigns pods (groups of containers) to nodes based on resource requirements, availability, and other scheduling policies.

5.1.3. Controller Manager

Manages various controllers that monitor the cluster state and take actions to maintain the desired state. Examples include the Node Controller, ReplicaSet Controller, and Deployment Controller.

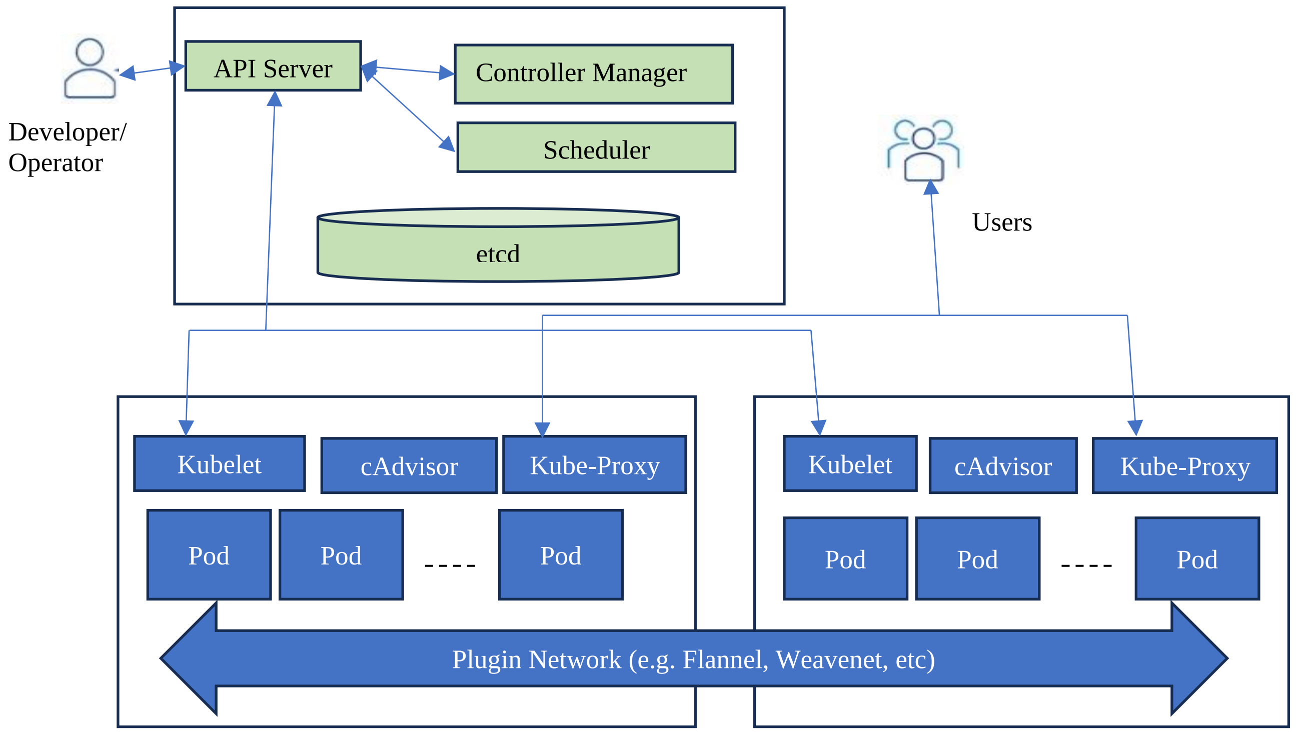

Fig. 2 Kubernetes architecture

5.1.4. etcd

A distributed key-value store that stores the cluster's configuration data and the state of the cluster.

5.2. Worker Nodes

Worker Nodes (also known as Minions or Worker Machines) are the worker units in the Kubernetes cluster. They run the actual containers that make up the applications. Each worker Node needs to have the following components:

5.2.1. Kubelet

The primary agent, running on each node, is responsible for managing and running containers as instructed by the Master Node.

5.2.2. Container Runtime

The software is responsible for running containers, such as Docker, containers, or CRI-O.

5.2.3. Kube-Proxy

Manages network connectivity and load balancing between services and pods on the node.

5.3. Pod

A Pod is the smallest deployable unit in Kubernetes. It represents a group of one or more containers that are scheduled and run together on the same node. Each Pod has its unique IP address and shares the same network namespace, allowing the containers within the Pod to communicate with each other using localhost. Pods are ephemeral and can be created, updated, or terminated independently.

5.4. ReplicaSet and Deployment

A ReplicaSet ensures that a specified number of identical Pods are running at all times. It helps with scaling and maintaining the desired number of replicas. A Deployment is a higher-level abstraction that manages

ReplicaSets. It allows for rolling updates, rollbacks, and other deployment strategies.

5.5. Service

Service provides a stable network endpoint to access a group of Pods. It abstracts the underlying Pod IP addresses and provides load balancing and service discovery.

5.6. Volume

A Volume is an abstraction that allows containers to store and access data. It provides data persistence and can be shared among multiple containers within a Pod.

5.7. Ingress

An Ingress is an API object that manages external access to services within a cluster. It provides routing rules, SSL termination, and load balancing for incoming traffic.

6. Kubernetes and Docker: Integration and Synergy

Kubernetes and Docker are two complementary technologies that work together to provide a powerful containerization and orchestration solution. Docker is a containerization platform that allows you to package applications and their dependencies into portable and isolated containers.

On the other hand, Kubernetes is a container orchestration platform that automates the deployment, scaling, and management of containerized applications across a cluster of nodes. Here is an overview of the integration and synergy between Kubernetes and Docker:

6.1. Container Runtime

Kubernetes is designed to be container-runtime agnostic, meaning it can work with different container runtimes. Docker is one of the most popular container runtimes supported by Kubernetes. Docker provides the underlying technology to build, run, and manage containers, while Kubernetes focuses on orchestrating and managing those containers at scale.

6.2. Container Images

Docker is widely used for building and managing container images. Kubernetes leverages Docker images as the basis for deploying applications within its cluster. Kubernetes pulls Docker images from container registries and deploys them as pods, the smallest deployable units within Kubernetes.

6.3. Container Deployment

Kubernetes leverages Docker containers to deploy applications across a cluster of worker nodes. Docker images, along with the application configuration, are defined in Kubernetes manifests (such as YAML files) and deployed using Kubernetes' declarative approach.

6.4. Container Lifecycle Management

Kubernetes provides advanced container lifecycle management features, such as scaling, rolling updates, and self-healing capabilities. Docker containers managed by Kubernetes can be automatically scaled up or down based on resource demands, allowing applications to adapt to varying workloads.

6.5. Container Networking and Storage

Kubernetes handles container networking and storage abstraction, providing services like load balancing, service discovery, and persistent volume management. Docker containers running within Kubernetes pods can communicate with each other using internal IP addresses, while Kubernetes manages the network routing and load balancing.

6.6. Container Orchestration

Kubernetes acts as a powerful container orchestration platform, managing the scheduling, placement, and scaling of Docker containers across a cluster of nodes. Kubernetes provides features like rolling updates, automated rollbacks, and fault tolerance, which enhance the reliability and availability of applications running in Docker containers.

6.7. Container Images

Docker is widely used for building and managing container images. Kubernetes leverages Docker images as the basis for deploying applications within its cluster. Kubernetes pulls Docker images from container registries and deploys them as pods, the smallest deployable units within Kubernetes.

6.8. Container Deployment

Kubernetes leverages Docker containers to deploy applications across a cluster of worker nodes. Docker images and the application configuration are defined in Kubernetes manifests (such as YAML files) and deployed using Kubernetes' declarative approach.

7. Use Cases

Kubernetes and Docker together offer a powerful combination for containerization and orchestration. Here are some common use cases for using Kubernetes and Docker together:

7.1. Application Deployment and Management

Use Docker to containerize your application and its dependencies, ensuring consistency and portability across different environments.

Deploy and manage your Docker containers using Kubernetes, which provides advanced features like automatic scaling, rolling updates, and self-healing capabilities. E.g., deploying and managing SAP HANA with Kubernetes and Docker offers benefits such as scalability, portability, and simplified management. By containerizing SAP HANA using Docker and deploying it in a Kubernetes cluster, organizations can leverage features like automatic scaling, high availability, and persistent storage.

7.2. Microservices Architecture

Adopt a microservices architecture by decomposing your application into smaller, loosely coupled services running in a separate Docker container. Use Kubernetes to orchestrate and manage the deployment of these microservices, allowing for scalability, fault tolerance, and easy service discovery. E.g., AWS offers a comprehensive ecosystem for building microservices architectures with Kubernetes and Docker. Organizations can easily manage and orchestrate containerized microservices deployments by leveraging Kubernetes on AWS.

Docker containers provide a standardized packaging format, ensuring consistency across different environments. AWS services like Amazon Elastic Kubernetes Service (EKS) simplify the deployment and management of Kubernetes clusters, offering scalability, automatic scaling, and high availability. Integration with AWS cloud-native services such as Amazon RDS for databases, Amazon S3 for object storage, and AWS Lambda for serverless computing enables efficient microservices communication and data storage. The combination of AWS, Kubernetes, and Docker provides a powerful solution for creating scalable and resilient microservices architectures in the cloud.

7.3. Hybrid and Multi-Cloud Deployments

Take advantage of Kubernetes' portability and flexibility to deploy applications across hybrid and multi-cloud environments. Use Docker to package your application into standardized containers easily deployed and managed by Kubernetes, regardless of the underlying infrastructure.

7.4. CI/CD Pipelines

Integrate Kubernetes and Docker into your CI/CD (Continuous Integration/Continuous Deployment) pipelines to automate the build, test, and deployment processes. Use Docker containers for consistent and reproducible builds and Kubernetes for deploying and testing applications in different environments. For e.g., both AWS and Azure offer robust CI/CD pipeline solutions that seamlessly integrate with Kubernetes and Docker, enabling efficient application deployment and continuous delivery. AWS Code Pipeline and Azure DevOps provide end-to-end automation for building, testing, and deploying containerized applications to Kubernetes clusters.

By leveraging Docker containers, developers can package their applications consistently, ensuring portability across different environments. Kubernetes acts as the orchestration layer, allowing for flexible scaling, rolling updates, and self-healing capabilities. With AWS and zure's CI/CD pipeline tools, developers can automate the entire release process, from code commits to deployment to production, while utilizing the power of Kubernetes and Docker to achieve reliable and efficient application delivery.

7.5. Resource Optimization

Utilize Kubernetes' container orchestration capabilities to optimize resource utilization across your infrastructure. Docker containers can be dynamically scheduled and scaled by Kubernetes based on resource demands, ensuring efficient resource allocation and cost optimization. For e.g., with SAP HANA, you can use Kubernetes and Docker to optimize resource utilization for your database workloads.

By containerizing SAP HANA using Docker, you can encapsulate the database software and configurations into portable and scalable containers. Kubernetes allows you to allocate resources to SAP HANA containers based on demand dynamically. You can configure resource limits and requests for CPU and memory to ensure efficient utilization. By monitoring SAP HANA's performance metrics and adjusting resource allocation accordingly, you can optimize compute resource usage and achieve better performance and cost efficiency.

7.6. Service Mesh and Networking

Combine Kubernetes and Docker with service mesh technologies like Istio or Linkerd for advanced networking, traffic management, and observability within your microservices architecture.

Docker containers running in Kubernetes can benefit from service mesh features like load balancing, traffic encryption, and fine-grained control over service-to-service communication.

8. Best Practices

Kubernetes and Docker together offer a powerful combination for containerization and orchestration. Here are some best practices for using Kubernetes and Docker together:

- Use a container registry to store and distribute your Docker images securely.
- Leverage Kubernetes' declarative approach by defining application deployments, services, and configurations in YAML manifests.
- Regularly update and patch your Docker images to address security vulnerabilities and ensure application reliability.
- Implement container security practices, such as image scanning, RBAC (Role-Based Access Control), and network policies in Kubernetes.
- Monitor and collect metrics from your Kubernetes cluster and Docker containers to gain insights into resource usage, performance, and application health.

9. Conclusion

The integration and synergy between Kubernetes and Docker provide a powerful solution for containerization and orchestration of applications. Docker simplifies the process of creating and managing container images, while Kubernetes offers advanced container orchestration capabilities for deploying, scaling, and managing containers at scale. By leveraging Docker for containerization and Kubernetes for orchestration, organizations can achieve portability, scalability, fault tolerance, and automation in their containerized environments. This integration enables the efficient management of microservices architectures, hybrid and multi-cloud deployments, CI/CD pipelines, resource optimization, and advanced networking capabilities. The combined use of Kubernetes and Docker empowers organizations to build and manage modern, scalable applications effectively while benefiting from the advantages of both technologies.

References

[1] George Whittaker, Kubernetes vs Docker: Exploring the Synergy in Containerization, 2023. [Online]. Available: https://www.linuxjournal.com/content/kubernetes-and-docker-exploring-synergy-containerization

[2] Kubernetes Documentation. [Online]. Available: https://kubernetes.io/docs/home/

[3] Docker Overview, 2023. [Online]. Available: https://docs.docker.com/get-started/overview/

[4] What is Kubernetes?, 2020. [Online]. Available: https://www.redhat.com/en/topics/containers/what-is-kubernetes

[5] What is Container Orchestration, 2022. [Online]. Available: https://www.redhat.com/en/topics/containers/what-is-container-orchestration

[6] Introduction to Container Orchestration: Kubernetes, Docker Swarm and Mesos with Marathon, 2016. [Online]. Available: https://www.exoscale.com/syslog/container-orchestration/

[7] Microservices Architecture. [Online]. Available: https://www.atlassian.com/microservices/microservices-architecture

[8] Container Orchestration with Kubernetes, 2023. [Online]. Available: https://www.xcubelabs.com/blog/container-orchestration-with-kubernetes/

[9] Orchestration. [Online]. Available: https://kubebyexample.com/learning-paths/container-fundamentals/introduction-containers/orchestration

[10] Anca Lordache, How Kubernetes Works Under the Hood with Docker Desktop. [Online]. Available: https://www.docker.com/blog/how-kubernetes-works-under-the-hood-with-docker-desktop/

[11] Don't Panic: Kubernetes and Docker, 2020. [Online]. Available: https://kubernetes.io/blog/2020/12/02/dont-panic-kubernetes-and-docker/

[12] Kubernetes vs Docker: Why Not Both?, 2022. [Online]. Available: https://www.ibm.com/blog/kubernetes-vs-docker/

[13] William Boyd, Kubernetes is Deprecating Docker: What you Need to Know, 2023. [Online]. Available: https://www.pluralsight.com/resources/blog/cloud/kubernetes-is-deprecating-docker-what-you-need-to-know

[14] Vivek Sonar, Running Kubernetes on AWS with EKS, 2021. [Online]. Available: https://www.airplane.dev/blog/running-kubernetes-on-aws-with-eks

[15] Brenden Burns, Empowering Developer Velocity and Efficiency with Kubernetes, 2020. [Online]. Available: https://azure.microsoft.com/en-us/blog/empowering-developer-velocity-and-efficiency-with-kubernetes/

[16] Brenden Burns, Build Resilient Applications with Kubernetes on Azure, 2020. [Online]. Available: https://azure.microsoft.com/en-us/blog/build-resilient-applications-with-kubernetes-on-azure/

[17] Kasper Siig, Running Kubernetes on Azure with AKS, 2021. [Online]. Available: [Online]. Available: https://www.airplane.dev/blog/running-kubernetes-on-azure-with-aks

[18] Serkan Ozal, Kubernetes CI/CD Pipelines Explained, 2022. [Online]. Available: https://thenewstack.io/kubernetes-ci-cd-pipelines-explained/

[19] CI/CD with Docker and Kubernetes, 2022. [Online]. Available: https://wpblog.semaphoreci.com/wp-content/uploads/2020/05/CICD_with_Docker_Kubernetes_Semaphore.pdf

[20] Harshit Mehndiratta, CI/CD Pipeline with Kubernetes, 2021. [Online]. Available: https://www.airplane.dev/blog/cicd-pipelines-with-kubernetes

[21] Denys van Kempen, At Your Service: SAP HANA in Containers | SAP HANA 2.0 – An Introduction, 2020. [Online]. Available: https://blogs.sap.com/2020/03/13/at-your-service-sap-hana-2.0-an-introduction-2/

[22] Sarath Chandra Dondapati, Proof of Concept: SAP on Kubernetes – Deployment, Application Scaling Scenarios, 2020. [Online]. Available: https://blogs.sap.com/2020/05/22/proof-of-concept-sap-on-kubernetes-deployment-application-and-database-scaling-scenarios/

[23] Maxim Afonin, Hana 2.0 Running Inside Docker, 2020. [Online]. Available: https://blogs.sap.com/2020/01/21/hana-2.0-running-inside-docker/