Kubernetes vs. Docker

What is Kubernetes?

Kubernetes

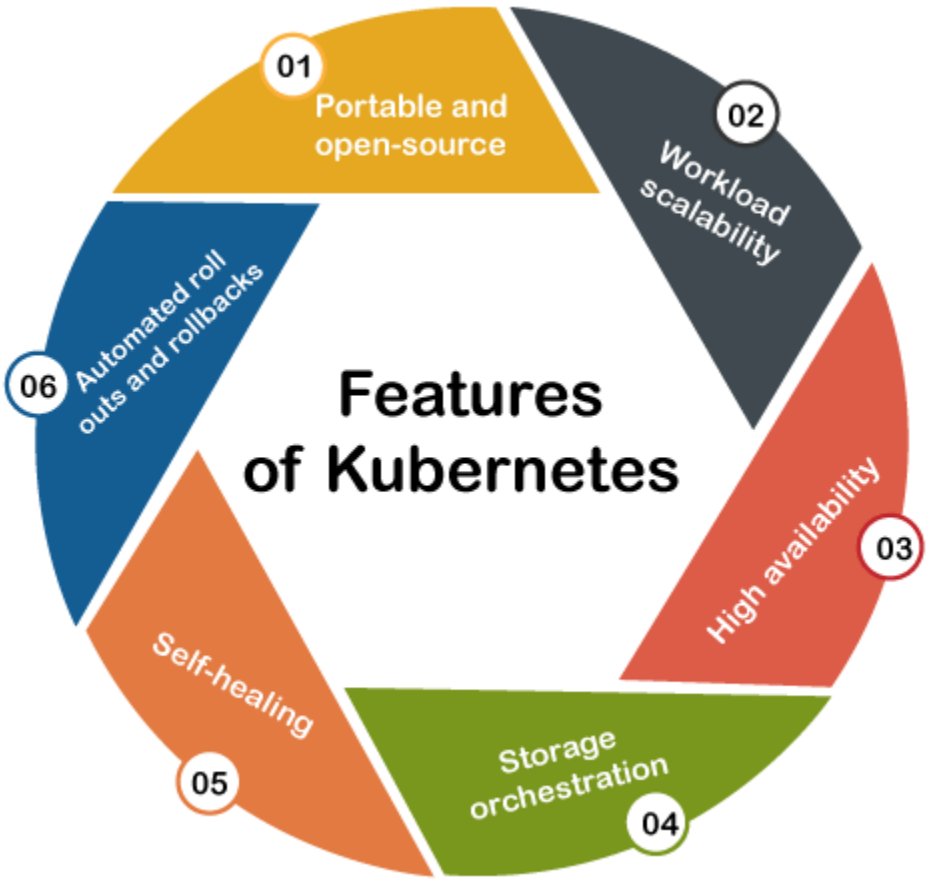
, being industrialized by Google, is an open-source container management software. In general, Kubernetes is used to manage, scale, and automate the deployment of containerized applications.

Containerized Applications can be understood as a process of enclosing an application by combining all of its files, libraries, and packages that are needed while running it on distinct platforms.

Kubernetes have become the best replacement for manual interaction. In the earlier times when we used to update all such containerized applications manually, we were required to conduct a regular cycle of processes: discontinuation of the older version, starting the new one, or rolling back to the previous one in case of error. Kubernetes have now overcome all such issues.

Features of Kubernetes

Kubernetes has the following features:

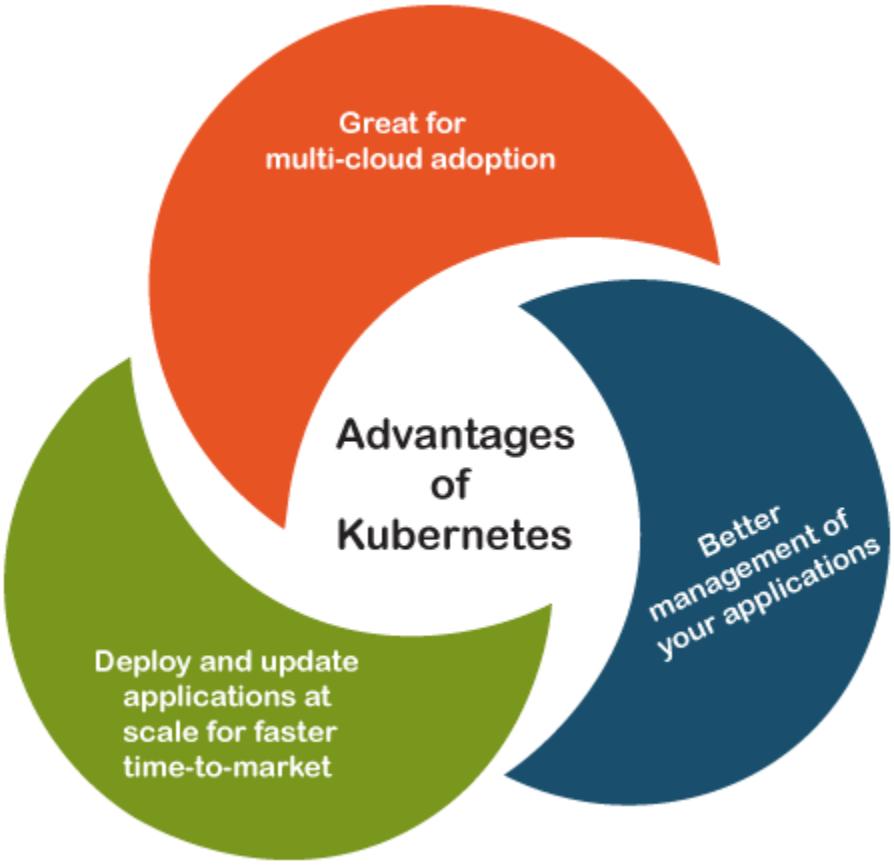


- **Portable and open-source**: Kubernetes is highly compatible crossways different Since it is open-source software, it provides greater flexibility.
- **Workload scalability**: Kubernetes is renowned for its systematic behavior. It can automatically add or remove new servers when required. In fact, it can manually scale as many containers as it can by simply changing the number of running containers.
- **High availability**: Kubernetes can easily cope up with the availability of both containers and infrastructure. Having said that, Kubernetes possesses reliability, which results in making it more accessible in any physical environment.

- **Storage orchestration**: Kubernetes is well known for its ability to organize storage systems such as local storage and public cloud providers.
- **Self-healing**: In case if the container fails to respond, the Kubernetes first try to restart it, and if they still don't reply on time, it either replaces the container or kills that container.
- **Automated roll-outs and rollbacks**: In general, Kubernetes defines the container's preferred state so as to transform the existing state into the preferred state at a controlled rate.

Advantages of Kubernetes

Following are the advantages of Kubernetes:



- **Great for multi-cloud adoption**
  In order to select the host as per the desired requirement, the microservice architecture

lets you efficiently fragment your application into smaller components encompassing containers that can be run on different cloud environments. Since Kubernetes is one of the best orchestration tools, you can easily use it anywhere. It can be deployed on any of the public/private/hybrid clouds, whereby you can actually reach out to the users placed at a distant location with greater availability and security. Also, it comes with "vendor lock-in" that facilitates avoiding potential hazards.

- **Deploy and update applications at scale for faster time-to-market** With the growth of modern software development, Kubernetes keeps pace with its regular demands because, otherwise, large teams will be forced to manually script their own deployment workflows.
- **Better management of your applications** Containers help in the breakdown of applications into smaller components so that they can be easily managed via Kubernetes, an orchestration tool. It helps in managing codebases and testing specific inputs and outputs.

Disadvantages of Kubernetes

Currently, Kubernetes is one of the best orchestration tools out there in the market. But everything is not perfect; it does have some demerits. So, the following are the disadvantages of the Kubernetes:

- Steep learning curve
- Quite difficult to migrate to stateless
- Offers limited functionality in comparison to Docker API
- Manual installation and configuration requires a lot of effort
- Absence of High Availability piece
- Not compatible with existing Docker CLI and Compose tools
- K8s talent is quite expensive

What is Docker?

Docker

is an OS-level virtualization software platform whereby the developers and IT administrators can create, deploy and run applications in a Docker Container with all their dependencies. Docker

Container is a lightweight software package that includes all the dependencies (frameworks, libraries, etc.) required to execute an application.

Docker is an open-source containerization platform that helps mechanize the deployment of an application as a lightweight container, which assists in the efficient working of an application in different environments.

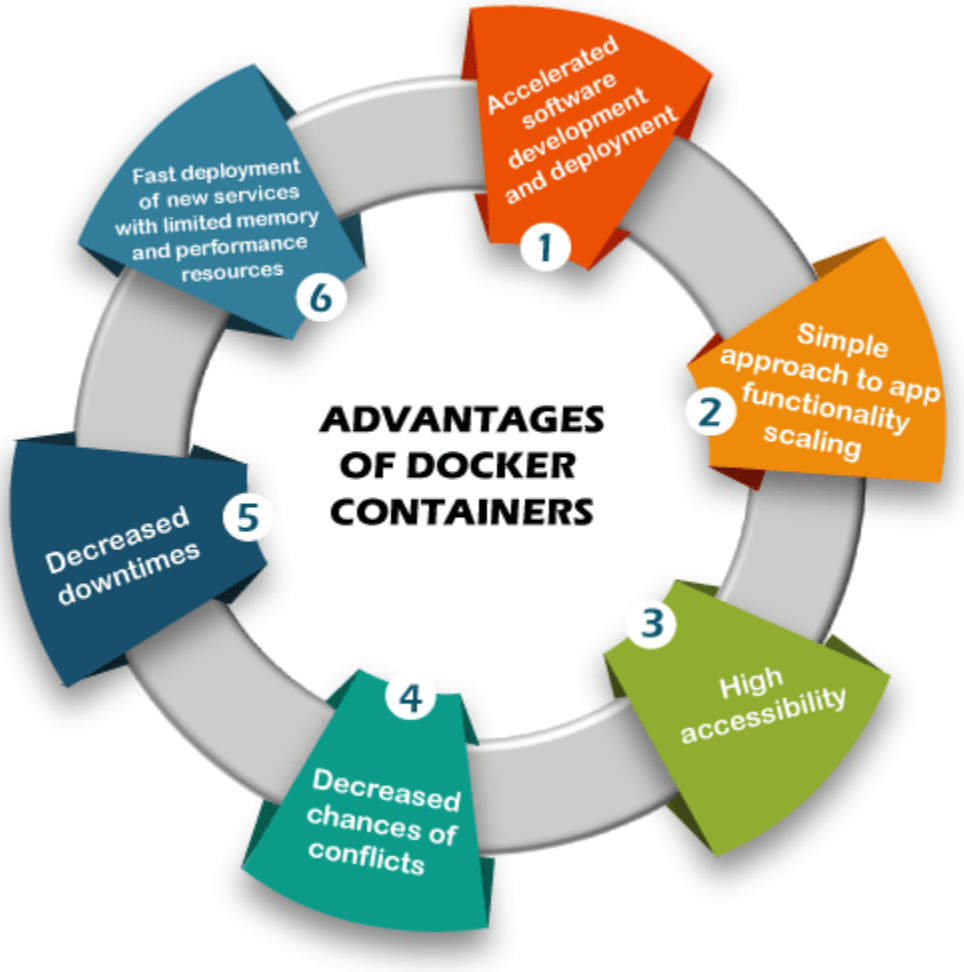
It is mainly contemplated as a solution that manages the isolated containers. Instead of creating resource-intensive virtual machines, Docker can efficiently put the software code in the containers so that it can be easily utilized in other systems. There are two ways to manage the software; either via a hybrid Cloud or by using the existing physical hardware resources.

In simple language, Docker Containers can be understood as virtualized abstract objects, which helps in connecting the code into dependencies (e.g., frameworks, config files, libraries, etc.). In general, Docker is used for building and deploying lightweight applications that are further installed on a single machine's (server or powerful computer) package. In comparison to traditional virtual machines, which are often called hypervisors, the containers improve performance by saving memory and consuming minimalist resources.

The applications created based on such containers employ hardware capabilities (including CPU, I/O devices, network devices, etc.) and do not require guest OSs. They mainly use the OS core of the hardware.

ADVANTAGES OF DOCKER CONTAINERS

Let's take a look at the main pros of Docker Containers:



- **Fast deployment of new services with limited memory and performance resources**: Since the Docker Containers are a lightweight alternate to hypervisors, they do not necessitate OS resources and the OS image's full load. The image can be launched from a private- or public-access Cloud, ensuring an instant launch of the software as long as minimum RAM stress load is available. Container not only offers high-speed computation but also necessitates less time to react to user requests.
- **Accelerated software development and deployment**: The significant use of Docker Containers can actually lower the consumption of resources that are required for the

creation and deployment of new software. Besides, it also alleviates the developer's burden of updating the prevailing network hardware (e.g., a more powerful server).

- **A simple approach to app functionality scaling**: In case you are looking to update the prevailing software, the Docker Container will ease your pain as it is mainly designed for migration purposes (i.e., you don't have to change a single line of code while scaling horizontally).
- **High accessibility**: Migration to Cloud allows the quick and easy transfer of an app to a new environment for further scaling or making the app business logic more complex. We can easily transfer the Docker Containers to any non-Cloud infrastructure. It also helps in evading the conflicts that have arisen in an OS with the installed software on the hardware level.
- **Decreased chances of conflicts**: The third-party applications, as well as their requirements, change in no time. If you are planning to integrate your Docker Containers-based software, you won't have any compatibility problems. This software solution has open-source code at its core, so it can be launched either on Linux and Windows OS, in the Cloud on virtual machines, etc.
- **Decreased downtimes**: The process of separating one Docker Container from the other containers is known as isolation. This allows for consistent performance and decreases downtime risks.

DISADVANTAGES OF DOCKER CONTAINERS

Among the most serious issues that might appear during the employment of the technology, developers point to the target software's vulnerability. For example, if you want to offer access to containers through web servers by utilizing an API, you will be required to thoroughly think through the parameter verification process. In general, you have to take care (e.g., with UNIX checking permissions) that no transformed data is passed in consort with a request by the hackers, which may provoke the generation of new containers.

Perhaps each orchestration tool carries a similar purpose, but there exist some fundamental differences like how the two operate. Following are some of the most important points that will help in differentiating Kubernetes from Dockers:

- **Application definition**

In Kubernetes, we can deploy applications by utilizing a group of pods, deployments, and services (or micro-services).

However, in the case of Docker Swarm, we can deploy applications as services (or micro-services) within a Swarm cluster. In order to specify a multi-container, we can use the YAML files, and for deploying the applications, we can use the Docker Compose.

- **Installation and set-up**

Kubernetes can be manually installed as it requires serious planning for keeping it up and running. Installation instructions may vary from OS-to-OS provider to provider. In Kubernetes, it is necessary to know the cluster configuration, like the IP addresses of a node or what role is taken by which node in advance.

However, the installation process of Docker Swarm is quite simple in contrast to Kubernetes. It only requires one set of tools to learn to build upon environment and configuration. Docker Swarm is more flexible as it permits new nodes to join a pre-existing cluster both as a manager or a worker.

- **Working on two systems**

In order to run Kubernetes on top of Docker, it is necessary to have prior knowledge about CLI (Command Line Interface). One must have hands-on Docker CLI to navigate inside a structure and supplemental Kubernetes common language infrastructure to run such programs.

Since Docker Swarm is itself a Docker tool, the same language is utilized to navigate within a structure. It provides variability as well as speeds up to the tool by offering a significant usability edge.

- **Logging and monitoring**

Kubernetes extends its supports to multiple versions of logging and monitoring as when the services get deployed within the cluster:

- Elasticsearch/Kibana (ELK) logs within the container
- Heapster/Grafana/ Influx for monitoring in the container
- Sysdig cloud integration

Whereas Docker Swarm can only support monitoring with third-party applications. In order to monitor using the Docker Swarm, it is highly suggested to use it with Reimann, and since Docker is an open API, it can be easily utilized to connect with several apps.

- **Scalability**

For distributed system, Kubernetes can be defined as an all-in-one framework. Since it offers a unified set of APIs, it is considered to be one of the complex systems which slow down the deployment and scaling of a container.

In contrast to Kubernetes, Docker Swarm, the deployment rate of containers are much faster, which results in quick scaling as per the demand.

- **High availability**

In order to provide high availability at the time of application failure, Kubernetes distributes all of its pods among nodes. In Kubernetes, load-balancing services can easily perceive unnatural pods and eradicate them, so it can be concluded that it supports high availability.

Docker Swarm also provides high availability as it can replicate the services in Swarm nodes. In Docker Swarm, Swarm manager nodes play a crucial role as they are the one who is responsible for the whole cluster and also it manages the resources of worker nodes.

- **Networking**

Since every individual pod can communicate with each other, so we can conclude that the Kubernetes network is flat as it also necessitates two CIDRs such that the first one requires pods to get an IP address and the other is for services.

In a Docker Swarm, a node connecting a cluster forms a network connection that spans all of the Swarm hosts and a host-only Docker bridge network for containers. In Docker Swarm, users can encrypt container data traffic when creating an overlay network on their own.