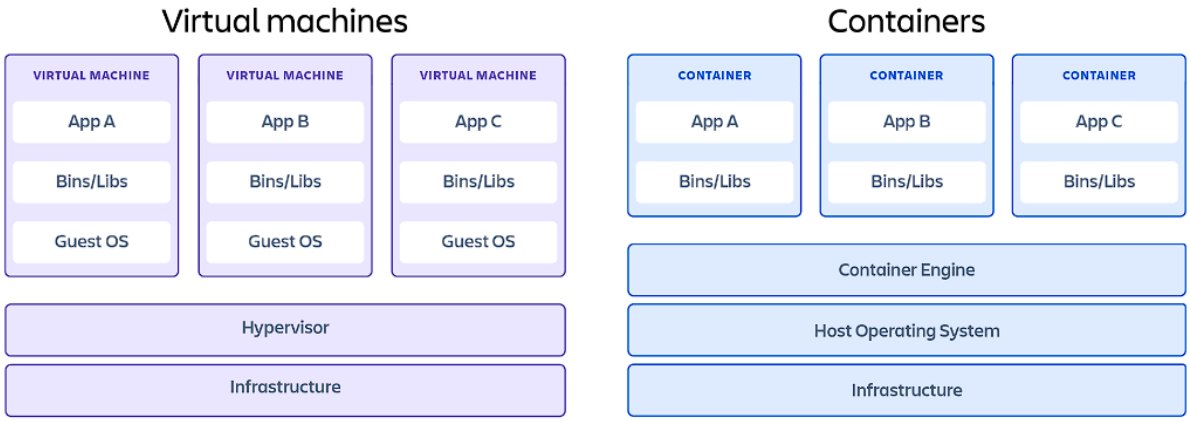
UNIT – 8 : INTRODUCTION TO DOCKER, CONTAINERS, KUBERNETES AND DEVOPS

Virtual Machine and Container:

Containers and virtual machines are very similar resource virtualization technologies. Virtualization is the process in which a system singular resource like RAM, CPU, Disk, or Networking can be 'virtualized' and represented as multiple resources. The key differentiator between containers and virtual machines is that virtual machines virtualize an entire machine down to the hardware layers and containers only virtualize software layers above the operating system level.



Virtual Machine (VM):

A VM can be defined as a virtual environment that works like a computer system (virtual) with its **CPU, storage, memory**, and **network interface** built on the physical hardware systems. A software known as hypervisor isolates the resources of the machine from hardware and arranges them properly.

Various physical machines that are equipped with any hypervisor like **KVM** (Kernel-based Virtual Machine) are known as a **host computer, host operating system, host machine,** or **simply host**. Many virtual machines that are using the resources are called a guest operating system, guest computers, guest machines, or simple guests.

A hypervisor treats all computer resources (such as CPU, storage, and memory) as the resource pool that can be relocated among new virtual machines or existing guests easily.

Virtual Machines allow more than one different types of operating system to execute on an individual computer simultaneously. All operating systems execute in the same fashion an application or operating system normally will on the host hardware.

Advantages of Virtual Machine:

A virtual machine can use more than one environment of the operating system on a similar computer system.

- Virtual machine facilitates the **ISA** (Instruction Set Architecture). The ISA structure is a different type of structure when compared to a real computer. It serves as an interface between hardware and software.
- Also, there are various security benefits for executing the VM. For example, when we need to execute a questionable security application, we can execute it in a guest OS. Thus, if an application leads to damage or loss, then it'll be temporary after a guest is shut down.
- The virtual machine permits for security forensics by auditing guest OS for defects and permitting all users to be quarantined for analysis.
- We build a hard disk (virtual) when we build our virtual machine. Hence, everything over that machine may crash, but if it happens, it will not affect any host machine.

Disadvantages of Virtual Machine:

- The virtual machine is not as efficient as actual machines due to virtual machines indirectly accessing the hardware.
- Running software over the top of a host OS means that the software will need to request access from the host. It will slow down the usability.
- If many virtual machines are executing on a similar host, performance might be decreased if the computer is executing with less sufficient power. In such a case, VM still uses our host machine's resources. The more capable the host computer system, the more immediately the VM will run.
- The virtual machine could be affected by the host machine's weaknesses. For example, the process isolation can be defined as an aspect which is employed by OS usually. But, there are some bugs that disrupt it. A common computer bare of the virtual machine will then affect only. However, a computer system along with lots of virtual machines will then also affect all of the machines.

Popular Virtual Machine Providers:

- **Virtualbox**: Virtualbox is a free and open source x86 architecture emulation system owned by Oracle. Virtualbox is one of the most popular and established virtual machine platforms with an ecosystem of supplementary tools to help develop and distribute virtual machine images.

- **VMware**: VMware is a publicly traded company that has built its business on one of the first x86 hardware virtualization technologies. VMware comes included with a hypervisor which is a utility that will deploy and manage multiple virtual machines. VMware has robust UI for managing virtual machines. VMware is a great enterprise virtual machine option offering support.
- **QEMU**: QEUM is the most robust hardware emulation virtual machine option. It has support for any generic hardware architecture. QEMU is a command line only utility and does not offer a graphical user interface for configuration or execution. This trade-off makes QEMU one of the fastest virtual machine options.

Container:

The container is a type of OS virtualization. An individual container could be used for running anything through a small software process or micro-service to a big application. Within a container, there are so many essential executables such as configuration **files, libraries,** and **binary code**.

Compared to machine or server virtualization approaches, a container does not include images of the operating system. It makes them portable and lightweight with fundamentally less overhead. More than one container can be deployed as multiple container clusters. These types of clusters can be handled by a container orchestrator like Kubernetes.

Container Advantages:

A container is a streamlined technique **to test, build, redeploy,** and **deploy** applications over more than one environment from a local laptop of the developer to a cloud or even the data center. The following are some important advantages of containers:

- **Less Overhead:** All containers need fewer resources of the system compared to hardware or traditional VM environments due to they do not contain images of an operating system.
- **Increased Portability:** Several applications executing inside the container could be deployed to more than one distinct hardware platforms and operating systems easily.
- **Consistent Operations:** Every team of DevOps knows that applications will execute the same without having to care where they're deployed in the containers.
- **Greater Efficiency:** Each container permits applications to be scaled, patched, and deployed more rapidly.
- **Application Development:** Every container support DevOps and Agile efforts for accelerating production, test, and development cycles.

Container Disadvantages:

- **Not good for every task:** Containers facilitate versatility; however, these are not a global substitute for each existing VM (virtual machine) deployment certainly. Also, a few applications are not a good option for container virtualization.
- **Overhead with dependencies:** General virtual machines are highly self-contained and all virtual machines contain a specific operating system, application components, and drivers. Bittman described that positioning a lot of dependencies over containers can limit the portability among various servers.
- **Weaker separation:** Container is weaker to share OS components and kernel. As a result, attacks and flaws have a greater chance of exploitation.
- **Limited tools:** The types of tools required to manage and monitor containers are lacking inside the industry. It is not a newer phenomenon. The previous days of hypervisor-based virtualization have been noticed by a lack of applicable tools.

Popular Container Providers:

- **Docker**: Docker is the most popular and widely used container runtime. Docker Hub is a giant public repository of popular containerized software applications. Containers on Docker Hub can instantly downloaded and deployed to a local Docker runtime.
- **RKT**: Pronounced "Rocket", RKT is a security-first focused container system. RKT containers do not allow insecure container functionality unless the user explicitly enables insecure features. RKT containers aim to address the underlying cross contamination exploitive security issues that other container runtime systems suffer from.
- **Linux Containers (LXC)**: The Linux Containers project is an open-source Linux container runtime system. LXC is used to isolate operating, system-level processes from each other. Docker actually uses LXC behind the scenes. Linux Containers aim to offer a vender neutral open-source container runtime.
- **CRI-O**: CRI-O is an implementation of the Kubernetes Container Runtime Interface (CRI) that allows the use of Open Container Initiative (OCI) compatible runtimes. It is a lightweight alternative to using Docker as the runtime for Kubernetes.

Difference Between VM and Container:

S.No.	Virtual Machines (VM)	Containers
1	VM is a piece of software that allows you to install other software inside of it so you control it virtually as opposed to installing the software directly on the computer.	While a container is software that allows different functionalities of an application independently.
2.	Applications running on a VM system, or hypervisor, can run different OS.	While applications running in a container environment share a single OS.
3.	VM virtualizes the computer system, meaning its hardware.	While containers virtualize the operating system, or the software only.
4.	VM size is very large, generally in gigabytes.	While the size of the container is very light, generally a few hundred megabytes, though it may vary as per use.
5.	VM takes longer to run than containers, the exact time depending on the underlying hardware.	While containers take far less time to run.
6.	VM uses a lot of system memory.	While containers require very less memory.
7.	VM is more secure, as the underlying hardware isn't shared between processes.	While containers are less secure, as the virtualization is software-based, and memory is shared.
8.	VMs are useful when we require all of the OS resources to run various applications.	While containers are useful when we are required to maximize the running applications using minimal servers.
9.	Examples of Type 1 hypervisors are KVM, Xen, and VMware. Virtualbox is a Type 2 hypervisor	Examples of containers are RancherOS, PhotonOS, and Containers by Docker.

Which option is better for you?

If you have specific hardware requirements for your project, or you are developing on one hardware platform and need to target another like Windows vs MacOS, you will need to use a virtual machine. Most other 'software only' requirements can be met by using containers.

How can you use containers and virtual machines together?

It is entirely possible to use containers and virtual machines in unison although the practical use-cases may be limited. A virtual machine can be created that emulates a unique hardware configuration. An operating system can then be installed within this virtual machine's hardware. Once the virtual machine is functional and boots the operating system, a container runtime can be installed on the operating system. At this point we have a functional computational system with emulated hardware that we can install containers on.

One practical use for this configuration is experimentation for system on chip deployments. Popular system on chip computational devices like the Raspberry Pi, or BeagleBone development boards can be emulated as a virtual machine, to experiment with running containers on them before testing on the actual hardware.

But the majority of the time, your needs will likely be met by one of the two. The key to deciding between containers or virtual machines for your virtualization needs is understanding your resource needs and the trade-offs you're willing to make.

Docker:

Docker is an open-source containerization platform by which you can pack your application and all its dependencies into a standardized unit called a container. Containers are light in weight which makes them portable and they are isolated from the underlying infrastructure and from each other container. You can run the docker image as a docker container in any machine where docker is installed without depending on the operating system.

What is a Dockerfile?

The Dockerfile uses DSL (Domain Specific Language) and contains instructions for generating a Docker image. Dockerfile will define the processes to quickly produce an image. While creating your application, you should create a Dockerfile in order since the **Docker daemon** runs all of the instructions from top to bottom.

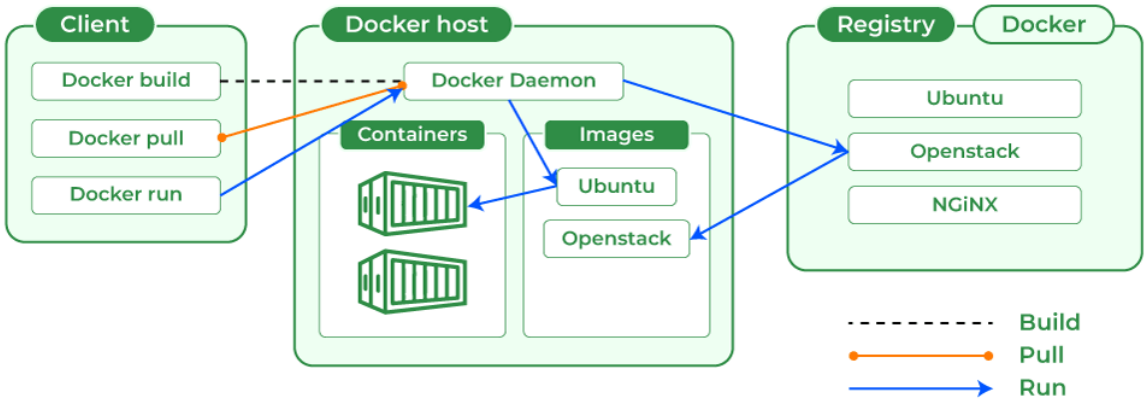
(The Docker daemon, often referred to simply as “Docker,” is a background service that manages Docker containers on a system.)

- It is a text document that contains necessary commands which on execution help assemble a Docker Image.
- Docker image is created using a Docker file.

Dockerfile is the source code of the image

How Docker Works?

Docker makes use of a client-server architecture. The Docker client talks with the docker daemon which helps in building, running, and distributing the docker containers. The Docker client runs with the daemon on the same system or we can connect the Docker client with the Docker daemon remotely. With the help of REST API over a UNIX socket or a network, the docker client and daemon interact with each other.



Components of a Docker Architecture:

1. **Docker Daemon (dockerd)**:
   - The Docker daemon is a persistent background process (daemon) that manages Docker objects such as images, containers, volumes, networks, and more.
   - It listens for Docker API requests and manages the Docker objects on the host system.
   - The daemon communicates with the Docker client and handles container orchestration tasks.
2. **Docker Client (docker)**:
   - The Docker client is a command-line interface (CLI) tool used to interact with the Docker daemon.

- It allows users to issue commands to the Docker daemon, such as building images, creating containers, managing networks, and inspecting Docker objects.
- The client communicates with the Docker daemon via the Docker Remote API.

3. **Docker Images**:
   - Docker images are read-only templates used to create containers.
   - They contain the application code, runtime, libraries, dependencies, and configuration files needed to run an application.
   - Images are built from Dockerfiles using the **docker build** command or pulled from Docker registries such as Docker Hub.
4. **Docker Containers**:
   - Docker containers are lightweight, runnable instances of Docker images.
   - They encapsulate the application and its dependencies into an isolated environment.
   - Containers can be started, stopped, moved, and deleted using Docker commands.
5. **Docker Registries**:
   - Docker registries are repositories that store Docker images.
   - They allow users to share, distribute, and collaborate on Docker images.
   - Docker Hub is a public Docker registry hosted by Docker, while private registries can be set up using Docker Trusted Registry (DTR) or third-party solutions.
6. **Docker Volumes**:
   - Docker volumes are persistent storage mechanisms used to persist data generated by containers.
   - They enable data sharing and persistence between containers and the host system.
   - Volumes can be managed and attached to containers using Docker commands.
7. **Docker Networks**:
   - Docker networks provide communication channels between containers and connect containers to each other and to external networks.
   - They facilitate network isolation, container discovery, and service connectivity.
   - Docker supports various network drivers for different use cases, such as bridge, overlay, and host networking.

8. **Docker Compose**:
    - Docker Compose is a tool used to define and run multi-container Docker applications.
    - It uses a YAML file (docker-compose.yml) to specify the services, networks, volumes, and other configurations required for a multi-container application.
    - Docker Compose simplifies the management of complex applications by orchestrating the deployment and scaling of multiple containers.

Key Features of Docker:

1. **Containerization**: Docker utilizes containerization technology to encapsulate applications and their dependencies into lightweight, portable containers. These containers are isolated from each other and from the underlying infrastructure, ensuring consistency and reproducibility across different environments.
2. **Efficiency**: Docker containers share the host system's kernel and resources, resulting in efficient resource utilization and minimal overhead compared to traditional virtualization methods. Containers start quickly and consume fewer system resources, making them ideal for deploying microservices and distributed applications.
3. **Portability**: Docker containers are platform-agnostic and can run on any system that supports Docker, including Linux, Windows, and macOS. This portability allows developers to build and test applications locally and deploy them seamlessly across different environments, from development to production.
4. **Scalability**: Docker provides tools for orchestrating and scaling containerized applications, such as Docker Swarm and Kubernetes. These tools enable automatic scaling, load balancing, and service discovery, allowing applications to handle increased traffic and workload demands efficiently.
5. **DevOps Integration**: Docker integrates seamlessly with DevOps tools and practices, facilitating continuous integration, continuous delivery (CI/CD), and infrastructure as code (IaC). Developers can use Docker to create reproducible build environments, automate testing and deployment workflows, and collaborate more effectively across teams.

Advantages of Docker:

- It runs the container in seconds instead of minutes.
- It uses less memory.

- It provides lightweight virtualization.
- It does not a require full operating system to run applications.
- It uses application dependencies to reduce the risk.
- Docker allows you to use a remote repository to share your container with others.
- It provides continuous deployment and testing environment.

Disadvantages of Docker:

- It increases complexity due to an additional layer.
- In Docker, it is difficult to manage large amount of containers.
- Some features such as container self -registration, containers self-inspects, copying files form host to the container, and more are missing in the Docker.
- Docker is not a good solution for applications that require rich graphical interface.
- Docker provides cross-platform compatibility means if an application is designed to run in a Docker container on Windows, then it can't run on Linux or vice versa.

Kubernetes (K8S):

Kubernetes is an open-source Container Management tool that automates container deployment, container scaling, descaling, and container load balancing (also called a container orchestration tool). It is written in Golang and has a vast community because it was first developed by Google and later donated to CNCF (Cloud Native Computing Foundation). Kubernetes can group 'n' number of containers into one logical unit for managing and deploying them easily. It works brilliantly with all cloud vendors i.e. public, hybrid, and on-premises.

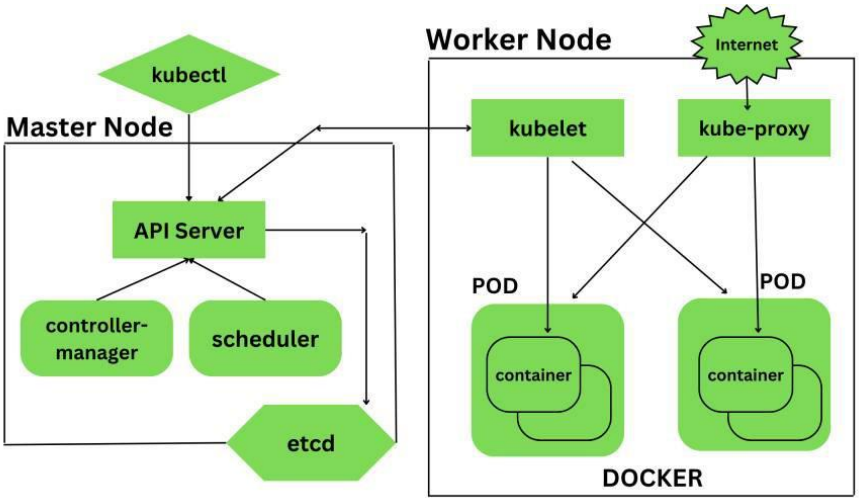
Features of Kubernetes:

1. **Automated Scheduling**– Kubernetes provides an advanced scheduler to launch containers on cluster nodes. It performs resource optimization.
2. **Self-Healing Capabilities**– It provides rescheduling, replacing, and restarting the containers that are dead.
3. **Automated Rollouts and Rollbacks**– It supports rollouts and rollbacks for the desired state of the containerized application.
4. **Horizontal Scaling and Load Balancing**– Kubernetes can scale up and scale down the application as per the requirements.

5. **Resource Utilization**– Kubernetes provides resource utilization monitoring and optimization, ensuring containers are using their resources efficiently.
6. **Support for multiple clouds and hybrid clouds**– Kubernetes can be deployed on different cloud platforms and run containerized applications across multiple clouds.
7. **Extensibility**– Kubernetes is very extensible and can be extended with custom plugins and controllers.
8. **Community Support-** Kubernetes has a large and active community with frequent updates, bug fixes, and new features being added.

Architecture of Kubernetes:

Kubernetes follows the client-server architecture where we have the master installed on one machine and the node on separate Linux machines. It follows the master-slave model, which uses a master to manage Docker containers across multiple Kubernetes nodes. A master and its controlled nodes (worker nodes) constitute a **"Kubernetes cluster"**. A developer can deploy an application in the docker containers with the assistance of the Kubernetes master.



Key Components of Kubernetes:

1. Kubernetes- Master Node Components

Kubernetes master is responsible for managing the entire cluster, coordinates all activities inside the cluster, and communicates with the worker nodes to keep the Kubernetes and your application running. This is the entry point of all administrative tasks. When we install Kubernetes on our system we have four primary components of Kubernetes Master that will get installed. The components of the Kubernetes Master node are:

API Server:

The API server is the entry point for all the REST commands used to control the cluster. All the administrative tasks are done by the API server within the master node. If we want to create, delete, update or display in Kubernetes object it has to go through this API server.API server validates and configures the API objects such as ports, services, replication, controllers, and deployments and it is responsible for exposing APIs for every operation. We can interact with these APIs using a tool called **kubectl**. *'kubectl' is a very tiny go language binary that basically talks to the API server to perform any operations that we issue from the command line. It is a command-line interface for running commands against Kubernetes clusters*

Scheduler:

It is a service in the master responsible for distributing the workload. It is responsible for tracking the utilization of the working load of each worker node and then placing the workload on which resources are available and can accept the workload. The scheduler is responsible for scheduling pods across available nodes depending on the constraints you mention in the configuration file it schedules these pods accordingly. The scheduler is responsible for workload utilization and allocating the pod to the new node.

Controller Manager:

Also known as controllers. It is a daemon that runs in a non terminating loop and is responsible for collecting and sending information to the API server. It regulates the Kubernetes cluster by performing lifestyle functions such as namespace creation and lifecycle event garbage collections, terminated pod garbage collection, cascading deleted garbage collection, node garbage collection, and many more. Basically, the controller watches the desired state of the cluster if the current state of the cluster does not meet the desired state then the control loop takes the corrective steps to make sure that the current state is the same as that of the desired state. The key controllers are the replication controller, endpoint controller, namespace controller, and service account, controller. So in this way controllers are responsible for the overall health of the entire cluster by ensuring that nodes are up and running all the time and correct pods are running as mentioned in the specs file.

etcd:

It is a distributed key-value lightweight database. In Kubernetes, it is a central database for storing the current cluster state at any point in time and is also used to store the configuration details such as subnets, config maps, etc. It is written in the Go programming language.

2. Kubernetes- Worker Node Components

Kubernetes Worker node contains all the necessary services to manage the networking between the containers, communicate with the master node, and assign resources to the containers scheduled. The components of the Kubernetes Worker node are:

Kubelet:

It is a primary node agent which communicates with the master node and executes on each worker node inside the cluster. It gets the pod specifications through the API server and executes the container associated with the pods and ensures that the containers described in the pods are running and healthy. If kubelet notices any issues with the pods running on the worker nodes then it tries to restart the pod on the same node. If the issue is with the worker node itself then the Kubernetes master node detects the node failure and decides to recreate the pods on the other healthy node.

Kube-Proxy:

It is the core networking component inside the Kubernetes cluster. It is responsible for maintaining the entire network configuration. Kube-Proxy maintains the distributed network across all the nodes, pods, and containers and exposes the services across the outside world. It acts as a network proxy and load balancer for a service on a single worker node and manages the network routing for TCP and UDP packets. It listens to the API server for each service endpoint creation and deletion so for each service endpoint it sets up the route so that you can reach it.

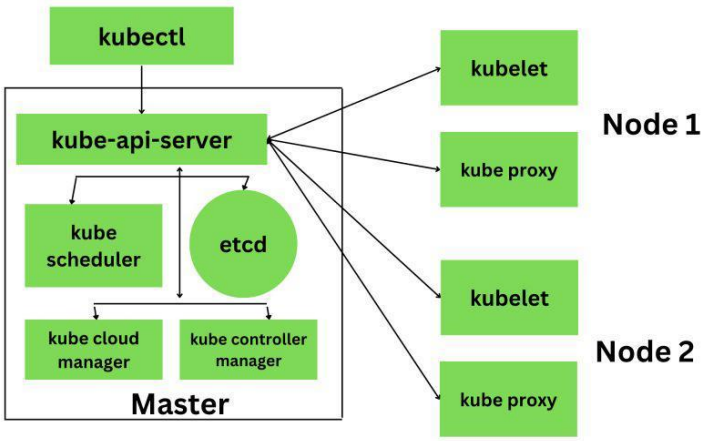
Pods:

A pod is a group of containers that are deployed together on the same host. With the help of pods, we can deploy multiple dependent containers together so it acts as a wrapper around these containers so we can interact and manage these containers primarily through pods.

Docker:

Docker is the containerization platform that is used to package your application and all its dependencies together in the form of containers to make sure that your application works seamlessly in any environment which can be development or test or production. Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Docker is the world's leading software container platform. It was launched in 2013 by a company called Dot cloud. It is written in the Go language. It has been just six years since Docker was launched yet communities have already shifted to it from VMs. Docker is designed to benefit both developers and system administrators making it a part of many DevOps toolchains. Developers can write code without worrying about the testing and production environment. Sysadmins need not worry about infrastructure as Docker can easily scale up and scale down the number of

systems. Docker comes into play at the deployment stage of the software development cycle.



Advantages of Kubernetes:

1. **Scalability**: Kubernetes allows applications to scale horizontally by adding or removing containers based on demand, ensuring optimal resource utilization and performance.
2. **Portability**: Kubernetes provides a consistent environment for deploying and running applications across different cloud providers, on-premises data centers, and developer workstations.
3. **High Availability**: Kubernetes automates the distribution and scheduling of containers across multiple nodes in a cluster, improving fault tolerance and ensuring high availability of applications.
4. **Self-Healing**: Kubernetes monitors the health of containers and automatically restarts or replaces them if they fail. It also supports rolling updates and can perform application upgrades with zero downtime.
5. **Extensibility**: Kubernetes has a modular architecture with a rich ecosystem of plugins and extensions, allowing users to customize and extend its functionality to suit their specific requirements.

Disadvantages of Kubernetes:

1. **Complexity**: Kubernetes has a steep learning curve and can be complex to set up, configure, and manage, especially for small teams or organizations with limited resources.

2. **Resource Overhead**: Running Kubernetes clusters requires additional infrastructure resources (compute, storage, networking) and may incur higher operational costs compared to simpler deployment solutions.
3. **Operational Complexity**: Operating Kubernetes clusters at scale requires expertise in areas such as networking, security, monitoring, and troubleshooting, which may pose challenges for organizations without dedicated DevOps teams.
4. **Tooling Fragmentation**: The Kubernetes ecosystem is vast and evolving rapidly, leading to fragmentation and compatibility issues between different tools, libraries, and distributions.

DevOps:

The DevOps is the combination of two words, one is **Development** and other is **Operations**. DevOps is a cultural and organizational approach that emphasizes collaboration, communication, integration, and automation between software development (Dev) and IT operations (Ops) teams. It aims to streamline the software delivery pipeline, improve deployment frequency, and enhance the quality and reliability of software applications. DevOps practices promote a shift from traditional siloed development and operations workflows to a more collaborative and agile approach.

Why DevOps?

- The operation and development team worked in complete isolation.
- After the design-build, the testing and deployment are performed respectively. That's why they consumed more time than actual build cycles.
- Without the use of DevOps, the team members are spending a large amount of time on designing, testing, and deploying instead of building the project.
- Manual code deployment leads to human errors in production.
- Coding and operation teams have their separate timelines and are not in synch, causing further delays.

How DevOps is different from Traditional IT?

Traditional IT has 1000s lines of code and is created by different teams with different standards whereas DevOps is created by one team with intimate knowledge of the product. Traditional IT is complex to understand and DevOps is easily understandable.

DevOps Lifecycle:

DevOps lifecycle is the methodology where professional development teams come together to bring products to market more efficiently and quickly. The structure of the DevOps lifecycle consists of Plan, Code, Building, Test, Releasing, Deploying, Operating, and Monitoring.

- **Plan:** Determining the commercial needs and gathering the opinions of end-user by professionals in this level of the DevOps lifecycle.
- **Code:** At this level, the code for the same is developed and in order to simplify the design, the team of developers uses tools and extensions that take care of security problems.
- **Build:** After the coding part, programmers use various tools for the submission of the code to the common code source.
- **Test:** This level is very important to assure software integrity. Various sorts of tests are done such as user acceptability testing, safety testing, speed testing, and many more.
- **Release:** At this level, everything is ready to be deployed in the operational environment.
- **Deploy:** In this level, Infrastructure-as-Code assists in creating the operational infrastructure and subsequently publishes the build using various DevOps lifecycle tools.
- **Operate:** At this level, the available version is ready for users to use. Here, the department looks after the server configuration and deployment.
- **Monitor:** The observation is done at this level that depends on the data which is gathered from consumer behavior, the efficiency of applications, and from various other sources.

Key Features of DevOps:

1. **Collaboration**:
   - DevOps encourages collaboration and cross-functional teamwork among developers, operations engineers, quality assurance (QA) professionals, and other stakeholders involved in the software delivery process.
   - Teams work together to break down organizational silos, share knowledge and responsibilities, and collectively drive towards common goals.
2. **Automation**:
   - Automation is a fundamental principle of DevOps, enabling the continuous integration (CI) and continuous delivery (CD) of software changes.
   - DevOps automation encompasses various tasks such as code compilation, testing, deployment, configuration management, infrastructure provisioning, and monitoring.
   - Automated workflows reduce manual errors, increase efficiency, and accelerate the pace of software delivery.
3. **Continuous Integration (CI)**:
   - CI is a DevOps practice that involves automatically integrating code changes into a shared repository multiple times a day.
   - Developers commit code changes to the repository frequently, triggering automated build and test processes to validate the changes.
   - CI helps identify integration issues early, maintain code quality, and enable faster feedback loops for developers.
4. **Continuous Delivery (CD)**:
   - CD extends CI by automating the release and deployment of code changes into production-like environments.
   - CD pipelines automate the steps required to build, test, and deploy applications across different environments, from development to production.
   - CD enables teams to release software updates quickly, reliably, and with minimal manual intervention.
5. **Infrastructure as Code (IaC)**:
   - IaC is a DevOps practice that involves managing and provisioning infrastructure resources (e.g., servers, networks, storage) using code and automation tools.

- Infrastructure configurations are defined in version-controlled templates or scripts, allowing infrastructure to be treated as code and managed alongside application code.
- IaC promotes consistency, repeatability, and scalability in infrastructure provisioning and management.

Advantages of DevOps:

- DevOps is an excellent approach for quick development and deployment of applications.
- It responds faster to the market changes to improve business growth.
- DevOps escalate business profit by decreasing software delivery time and transportation costs.
- DevOps clears the descriptive process, which gives clarity on product development and delivery.
- It improves customer experience and satisfaction.
- DevOps simplifies collaboration and places all tools in the cloud for customers to access.
- DevOps means collective responsibility, which leads to better team engagement and productivity.

Disadvantages of DevOps:

- DevOps professional or expert's developers are less available.
- Developing with DevOps is so expensive.
- Adopting new DevOps technology into the industries is hard to manage in short time.
- Lack of DevOps knowledge can be a problem in the continuous integration of automation projects.