# DOCKER /CONTAINER

**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's container. You can run the docker image as a docker container in any machine where docker is installed without depending on the operating system.

There are two big pieces to Docker: The **Docker Engine**, which is the Docker binary that's running on your local machine and servers and does the work to run your software. The **Docker Hub** is a website and cloud service that makes it easy for everyone to share their docker images.

**Docker is Popular**

Docker gained its popularity due to its impact on the software development and deployment. The following are the some of the main reasons for docker becoming popular:

1. **Portability:** Docker facilitates the developers in packaging their applications with all dependencies into a single lightweight containers. It facilities in ensuring the consistent performance across the different computing environments.
2. **Reproducibility:** Encapsulating the applications with their dependencies within a container it ensures in software setups remaining consistent across the development, testing and production environments.
3. **Efficiency:** Docker, through its container-based architecture it optimizes the resource utilization. It allows the developers to run the multiple isolated applications on a single host system.
4. **Scalability:** Docker's scalability features facilitated the developers in making easier of their applications handling at time of workloads increment.

> *Understanding Docker's core concepts is essential, but practical experience is what truly sets you apart. Platforms like Hostinger make it easy to deploy Docker containers, allowing you to focus on developing and testing your applications. Hostinger's* ***scalable infrastructure*** *provides an ideal environment for learning and*

*experimenting with Docker in a real-world setting. Their **seamless integration with Docker containers** ensures that whether you're running simple apps or complex multi-container setups, you can deploy with ease.*

**Key Components of Docker**

The following are the some of the key components of Docker:

- **Docker Engine:** Docker Engine is a core part of docker, that handles the creation and management of containers.
- **Docker Image:** Docker Image is a read-only template that is used for creating containers, containing the application code and dependencies.
- **Docker Hub:** It is a cloud based repository that is used for finding and sharing the container images.
- **Dockerfile:** It is a file that describes the steps to create an image quickly.
- **Docker Registry** : It is a storage distribution system for docker images, where you can store the images in both public and private modes.

**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 Dockerfile is the source code of the image.

(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 Dockerfile.


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. To know more about working of docker refer to the Architecture of Docker .

**Docker Image**

It is a file, comprised of multiple layers, used to execute code in a Docker container. They are a set of instructions used to create docker containers. Docker Image is an executable package of software that includes everything needed to run an application.

This image informs how a container should instantiate, determining which software components will run and how. Docker Container is a virtual environment that bundles application code with all the dependencies required to run the application. The application runs quickly and reliably from one computing environment to another.

**Docker Container?**

Docker container is a runtime instance of an image. Allows developers to package applications with all parts needed such as libraries and other dependencies. Docker Containers are runtime instances of Docker images. Containers contain the whole kit required for an application, so the application can be run in an isolated way.

> *For eg.- Suppose there is an image of Ubuntu OS with NGINX SERVER when this image is run with the docker run command, then a container will be created and NGINX SERVER will be running on Ubuntu OS.*

**Docker Hub?**

Docker Hub is a repository service and it is a cloud-based service where people push their Docker Container Images and also pull the Docker Container Images from the Docker Hub anytime or anywhere via the internet. Generally it makes it easy to find and reuse images. It provides features such as you can push your images as private or public registry where you can store and share Docker images.

Mainly DevOps team uses the Docker Hub. It is an open-source tool and freely available for all operating systems. It is like storage where we store the images and

pull the images when it is required. When a person wants to push/pull images from the Docker Hub they must have a basic knowledge of Docker. Let us discuss the requirements of the Docker tool.


**Docker Commands**

Through introducing the essential docker commands, docker became a powerful software in streamlining the container management process. It helps in ensuring a seamless development and deployment workflows. The following are the some of docker commands that are used commonly:

- **Docker Run:** It used for launching the containers from images, with specifying the runtime options and commands.
- **Docker Pull:** It fetches the container images from the container registry like Docker Hub to the local machine.
- **Docker ps** : It helps in displaying the running containers along with their important information like container ID, image used and status.
- **Docker Stop** : It helps in halting the running containers gracefully shutting down the processes within them.
- **Docker Start:** It helps in restarting the stopped containers, resuming their operations from the previous state.
- **Docker Login:** It helps to login in to the docker registry enabling the access to private repositories.

> *To Know more about the docker commands refer tot the Docker – Instruction Commands*

**Docker Engine**

The software that hosts the containers is named Docker Engine. Docker Engine is a client-server based application. The docker engine has 3 main components:

1. **Server:** It is responsible for creating and managing Docker images, containers, networks, and volumes on the Docker. It is referred to as a daemon process.
2. REST API **:** It specifies how the applications can interact with the Server and instructs it what to do.

3. **Client:** The Client is a docker command-line interface (CLI), that allows us to interact with Docker using the docker commands.

**Difference Between Docker Containers and Virtual Machines**

The following are the differences between docker containers and Virtual Machines:

| Docker Containers | Virtual Machines |
|---|---|
| Docker Containers contain binaries, libraries, and configuration files along with the application itself. | Virtual Machines (VMs) run on Hypervisors, which allow multiple Virtual Machines to run on a single machine along with its own operating system. |
| They don't contain a guest OS for each container and rely on the underlying OS kernel, which makes the containers lightweight. | Each VM has its own copy of an operating system along with the application and necessary binaries, which makes it significantly larger and it requires more resources. |
| Containers share resources with other containers in the same host OS and provide OS-level process isolation. | They provide Hardware-level process isolation and are slow to boot. |

**Importance of Docker**

The following are the some of the insights that discusses on the importance of docker:

- **Efficiency and Speed** : It facilitates with providing the streamlined development and deployment by packaging applications with dependencies into consistent containers.
- **Resource Optimization** : It helps in sharing host system resources efficiently, allowing for higher application density and cost savings.

- **Scalability and Portability** : It is easily able to scale the applications and ensures seamless movement across different environments.
- **Isolation and Security** : it provides high isolation, reducing conflicts and enhancing security.

**Benefits of Docker**

The following are the some of the benefits of Docker:

- **Portability:** Docker facilities with creation of lightweight portable containers that can be unable on any machine regardless of the underlying operating systems.
- **Isolation:** Docker through containers provides a high level of isolation with enabling the applications to run independently of each other addressing the issues that one container doesn't impact on other.
- **Reproducibility:** With, Docker developers can easily package their applications and their dependencies into a reusable images. It allows for consistent and repoduciable builds across the development, testing and production environments.
- **DevOps Integration** : It promotes the collaboration and automation across the software development life cycle in handing the increasing workloads.

**Use Cases of Docker**

The following are the some of the use cases of Docker:

**1. Continuous Integration and Continuous Deployment (CI/CD)** : It helps in streamlining and automating the software delivery process with ensuring faster and more reliable releases.

**Example:** A team working on an online store can use Docker to automatically test and update the website every time someone makes a change to the code.

**2. Microservices Architecture:** It facilitates the development, deployment, and management of microservices with enabling independent scaling and maintenance.

**Example:** A food delivery app can have separate containers for logging in, placing orders, and tracking deliveries. Each part can be updated or fixed without affecting the rest.

**3. Development Environment Consistency:** Docker gives all developers the same environment to work in, no matter what computer they use.

**Example:** In a team one person is using Windows and another may use Mac, but Docker makes sure the app runs the same way for everyone, avoiding bugs like "it works on my machine and not working in other system".

**4. Multi-Cloud and Hybrid Cloud Deployments:** Docker makes it simple to run your applications on different cloud platforms like AWS, Azure, or Google Cloud.

**Example:** A business using AWS today can move their app to Google Cloud tomorrow with little effort, with the help of Docker containers.

**Docker V/s Kubernetes**

The following are the difference between docker and kubernetes:

| Feature | Docker | Kubernetes |
|---|---|---|
| **Primary Purpose** | Containerization platform | Container orchestration platform |
| **Functionality** | Creates and manages containers | Manages and scales containerized applications |
| **Setup Complexity** | Simple to set up and use | More complex setup and configuration |
| **Scalability** | Limited to single-node scaling | Designed for large-scale, multi-node environments |
| **Networking** | Basic networking capabilities | Advanced networking with service discovery and load balancing |
| **State Management** | Stateless; manages individual containers | Stateful; manages container clusters and services |
| **Use Case** | Development and testing environments | Production environments with high scalability and reliability requirements |

**Importance of Docker**

The following are the some of the insights that discusses on the importance of docker:

- **Efficiency and Speed** : It facilitates with providing the streamlined development and deployment by packaging applications with dependencies into consistent containers.
- **Resource Optimization** : It helps in sharing host system resources efficiently, allowing for higher application density and cost savings.
- **Scalability and Portability** : It is easily able to scale the applications and ensures seamless movement across different environments.
- **Isolation and Security** : it provides high isolation, reducing conflicts and enhancing security.

**Benefits of Docker**

The following are the some of the benefits of Docker:

- **Portability:** Docker facilities with creation of lightweight portable containers that can be unable on any machine regardless of the underlying operating systems.
- **Isolation:** Docker through containers provides a high level of isolation with enabling the applications to run independently of each other addressing the issues that one container doesn't impact on other.
- **Reproducibility:** With, Docker developers can easily package their applications and their dependencies into a reusable images. It allows for consistent and repoduciable builds across the development, testing and production environments.
- **DevOps Integration** : It promotes the collaboration and automation across the software development life cycle in handing the increasing workloads.

**Use Cases of Docker**

The following are the some of the use cases of Docker:

**1. Continuous Integration and Continuous Deployment (CI/CD)** : It helps in streamlining and automating the software delivery process with ensuring faster and more reliable releases.

**Example:** A team working on an online store can use Docker to automatically test and update the website every time someone makes a change to the code.

**2. Microservices Architecture:** It facilitates the development, deployment, and management of microservices with enabling independent scaling and maintenance.

**Example:** A food delivery app can have separate containers for logging in, placing orders, and tracking deliveries. Each part can be updated or fixed without affecting the rest.

**3. Development Environment Consistency:** Docker gives all developers the same environment to work in, no matter what computer they use.

**Example:** In a team one person is using Windows and another may use Mac, but Docker makes sure the app runs the same way for everyone, avoiding bugs like "it works on my machine and not working in other system".

**4. Multi-Cloud and Hybrid Cloud Deployments:** Docker makes it simple to run your applications on different cloud platforms like AWS, Azure, or Google Cloud.

**Example:** A business using AWS today can move their app to Google Cloud tomorrow with little effort, with the help of Docker containers.

**Docker V/s Kubernetes**

The following are the difference between docker and kubernetes:

| Feature | Docker | Kubernetes |
|---|---|---|
| **Primary Purpose** | Containerization platform | Container orchestration platform |
| **Functionality** | Creates and manages containers | Manages and scales containerized applications |
| **Setup Complexity** | Simple to set up and use | More complex setup and configuration |
| **Scalability** | Limited to single-node scaling | Designed for large-scale, multi-node environments |
| **Networking** | Basic networking | Advanced networking with service |

| Feature | Docker | Kubernetes |
| --- | --- | --- |
| | capabilities | discovery and load balancing |
| **State Management** | Stateless; manages individual containers | Stateful; manages container clusters and services |
| **Use Case** | Development and testing environments | Production environments with high scalability and reliability requirements |