# What is Docker?

Docker is an open-source platform that enables developers to build, deploy, run, update and manage containers.

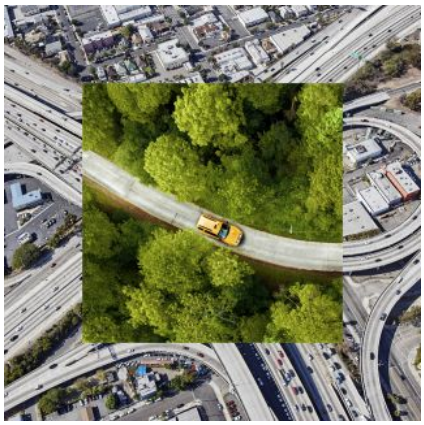
Containers are standardized, executable components that combine application source code with the operating system (OS) libraries and dependencies required to run that code in any environment.

It's worth noting that when people talk about Docker, they are usually referring to Docker Engine, the runtime for building and running containers. Docker also refers to Docker, Inc.[1], the company that sells the commercial version of Docker. It also relates to the Docker open source project[2], to which Docker, Inc. and many other organizations and individuals contribute.

Containers simplify the development and delivery of distributed applications. They have become increasingly popular as organizations shift to cloud-native development and hybrid multicloud environments. Developers can create containers without Docker by working directly with capabilities built into Linux® and other operating systems, but Docker makes containerization faster and easier. As of this writing, Docker reported that over 20 million developers use the platform monthly[3]. Like other containerization technologies, including Kubernetes, Docker plays a crucial role in modern software development, specifically microservices architecture.

## What are microservices?

In contrast to the traditional monolithic approach of a large, tightly coupled application, microservices provide a cloud-native architectural framework. This framework composes a single application from many smaller, loosely coupled and independently deployable smaller components or services. Each service can be containerized using Docker, simplifying deployment and allowing teams to roll out new versions and scale applications as needed.





## How do containers work?

Containers are made possible by process isolation and virtualization capabilities built into the Linux kernel. These capabilities include control groups (Cgroups) for allocating resources among processes and namespaces for restricting a process's access or visibility into other resources or areas of the system.

Containers enable multiple application components to share the resources of a single instance of the host operating system. This sharing is similar to how a hypervisor allows multiple virtual machines (VMs) to share a single hardware server's central processing unit (CPU), memory and other resources.

Container technology offers all the functions and benefits of virtual machines (VMs)—including application isolation and cost-effective scalability, plus significant other advantages:

**Lighter weight:** Unlike VMs, containers don't carry the payload of an entire OS instance and hypervisor. They include just the OS processes and dependencies necessary to run the code. Container sizes are measured in megabytes (versus gigabytes for some VMs), make better use of hardware capacity and have faster startup times.

**Improved productivity:** Containerized applications can be written once and run anywhere. Compared to VMs, containers are quicker and easier to deploy, provision and restart.

**Greater efficiency:** With containers, developers can run several times as many copies of an application on the same hardware as they can using VMs. This efficiency can reduce cloud spending.

Organizations that use containers report many other benefits including improved app quality, faster response to market changes and more.

**AI Academy**

Achieving AI-readiness with hybrid cloud

Led by top IBM thought leaders, the curriculum is designed to help business leaders gain the knowledge needed to prioritize the AI investments that can drive growth.

Go to episode →

## Why use Docker and why is it so popular?

Docker is the most widely used containerization tool, with an 82.84% market share.[4] Docker is so popular today that "Docker" and "containers" are used interchangeably. However, the first container-related technologies were available for years—even decades[5]—before Docker was publicly released as open source in 2013.

Most notably, in 2008, LinuXContainers (LXC) was implemented in the Linux kernel, fully enabling virtualization for a single instance of Linux. While LXC is still used today, newer Linux kernel technologies are available. Ubuntu, a modern, open source Linux operating system, also provides this capability. Docker lets developers access these native containerization capabilities by using simple commands and automate them through a work-saving application programming interface (API).

## Docker versus LXC

There are many advantages to using Docker over the older, less compatible LXC.

**Improved and seamless container portability**
While LXC containers often reference machine-specific configurations, Docker containers run without modification across any desktop, data center or cloud computing environment.

**Even lighter weight and more granular updates**
With Docker, developers can combine multiple processes within a single container. This flexibility makes it possible to build an application that can continue running while one of its parts is taken down for an update or repair.

**Automated container creation**
Docker can automatically build a container based on application source code.

**Container versioning**
Docker can track versions of a container image, roll back to previous versions and trace who built a version and how. Docker can even upload only the deltas (partial releases of software modules) between an existing version and a new one.

**Container reuse**
Developers can access an open source registry containing thousands of user-contributed containers.

Today, Docker containerization also works with Microsoft Windows Server and Apple MacOS. Developers can run Docker containers on any operating system. All of the leading cloud service providers (CSPs), including Amazon Web Services (AWS), Microsoft Azure, Google Cloud Services and IBM Cloud®, offer specific services to help developers build, deploy and run applications containerized with Docker.

## Docker architecture, terms and tools

Docker uses a client/server architecture. The following is a breakdown of the core components associated with Docker, along with other Docker terms and tools.

**Docker host**: A Docker host is a physical or virtual machine running Linux (or another Docker-Engine compatible OS).

**Docker Engine:** Docker engine is a client/server application consisting of the Docker daemon, a Docker API that interacts with the daemon, and a command-line interface (CLI) that talks to the daemon.

**Docker daemon:** Docker daemon is a service that creates and manages Docker images, by using the commands from the client. Essentially the Docker daemon serves as the control center for Docker implementation.

**Docker client:** The Docker client provides the CLI that accesses the Docker API (a REST API) to communicate with the Docker daemon over Unix sockets or a network interface. The client can be connected to a daemon remotely, or a developer can run the daemon and client on the same computer system.

**Docker objects:** Docker objects are components of a Docker deployment that help package and distribute applications. They include images, containers, networks, volumes, plug-ins and more.

**Docker containers:** Docker containers are the live, running instances of Docker images. While Docker images are read-only files, containers are live, ephemeral, executable content. Users can interact with them, and administrators can adjust their settings and conditions by using Docker commands.

**Docker images:** Docker images contain executable application source code and all the tools, libraries and dependencies the application code needs to run as a container. When a developer runs the Docker image, it becomes one instance (or multiple instances) of the container.

Building Docker images from scratch is possible, but most developers pull them down from common repositories. Developers can create multiple Docker images from a single base image and will share their stack's commonalities.

Docker images are made up of layers, and each layer corresponds to a version of the image. Whenever a developer makes changes to an image, a new top layer is created, and this top layer replaces the previous top layer as the current version of the image. Previous layers are saved for rollbacks or to be reused in other projects.

Each time a container is created from a Docker image, yet another new layer called the container layer is created. Changes made to the container—like adding or deleting files—are saved to the container layer, and these changes only exist while the container is running.

This iterative image-creation process increases overall efficiency since multiple live container instances can run from a single base image. When they do so, they use a common stack.

**Docker build**: Docker build is a command that has tools and features for building Docker images.

**Dockerfile:** Every Docker container starts with a simple text file containing instructions for how to build the Docker container image. Dockerfile automates the process of creating Docker images. It's essentially a list of CLI instructions that Docker Engine will run to assemble the image. The list of Docker commands is vast but standardized: Docker operations work the same regardless of contents, infrastructure or other environment variables.

**Docker documentation**: Docker documentation, or Docker docs, refers to the official Docker library of resources, manuals and guides for building containerized applications.

**Docker Hub**: Docker Hub[6] is the public repository of Docker images, calling itself the world's largest library and community for container images[7]. It holds over 100,000 container images sourced from commercial software vendors, open source projects and individual developers. Docker Hub includes images produced by Docker, Inc., certified images belonging to the Docker Trusted Registry and thousands of other images.

All Docker Hub users can share their images at will. They can also download predefined base images from the Docker filesystem as a starting point for any containerization project.

Other image repositories exist, including GitHub[8]. GitHub is a repository hosting service well known for application development tools and as a platform that fosters collaboration and communication. Users of Docker Hub can create a repository (repo) that can hold many images. The repository can be public or private and linked to GitHub or BitBucket accounts.

**Docker Desktop:** Docker Desktop is an application for Mac or Windows that includes Docker Engine, Docker CLI client, Docker Compose, Kubernetes and others. It also provides access to Docker Hub.

**Docker registry:** A Docker registry is a scalable, open-source storage and distribution system for Docker images. It enables developers to track image versions in repositories by using tagging for identification. This tracking and identification are accomplished by using Git, a version control tool.

**Docker plug-ins:** Developers use plug-ins to make Docker Engine even more functional. Several Docker plugins supporting authorization, volume and network are included in the Docker Engine plug-in system; third-party plug-ins can be loaded as well.

**Docker extensions:** Docker extensions enable developers to use third-party tools within Docker Desktop to extend its functions. Extensions for developer tools include Kubernetes app development, security, observability and more.

**Docker Compose:** Developers can use Docker Compose to manage multicontainer applications, where all containers run on the same Docker host. Docker Compose creates a YAML (.YML) file that specifies which services are included in the application and can deploy and run containers with a single command. Because YAML syntax is language-agnostic, YAML files can be used in programs written in Java, Python, Ruby and many other languages.

Developers can also use Docker Compose to define persistent volumes for storage, specify base nodes and document and configure service dependencies.

## Kubernetes and container orchestration

When running just a few containers, managing an application within Docker Engine, the industry's de facto runtime, is pretty simple. However, for deployments comprising thousands of containers and hundreds of services, it's nearly impossible to monitor and manage container lifecycles without a container orchestration tool.

While Docker includes its own orchestration tool (called Docker Swarm), Kubernetes is the industry standard. Other popular container orchestration platforms include Apache Mesos and Nomad.

Kubernetes is an open source container orchestration platform descended from Borg, a project developed for internal use at Google. Introduced to the public in 2014, Kubernetes schedules and automates tasks integral to managing container-based architectures, including container deployment, updates, service discovery, storage provisioning, load balancing, health monitoring and more. In 2015, Google donated Kubernetes to the Cloud Native Computing Foundation (CNCF)[9], the open source, vendor-neutral hub of cloud-native computing.

In addition, the open source ecosystem of tools for Kubernetes, including Istio, Knative and Tekton, enables organizations to deploy a high-productivity platform as a service (PaaS) for containerized applications. This ecosystem also provides a faster on-ramp to serverless computing.

Kubernetes Explained (10:59 min)

## Docker and container industry standards

Founded in 2015 after Docker donated the container image specification and runtime code runc, the Open Container Initiative (OCI)[10] is a Linux Foundation project committed to creating open industry standards around the container image format and runtime. The OCI consists of leading

companies, including Docker, IBM and Red Hat®. It supports innovation while helping organizations avoid vendor lock-in.

While Docker is the most well-known and highly used container technology, the broader ecosystem has standardized on containerd and other alternatives, including LXC, CRI-O, Podman and others. In 2017, Docker donated the containerd project to the CNCF. Containerd is an industry-standard container runtime that uses runc and is the core container runtime of the Docker Engine.

## Docker security

The complexity surrounding containerized workloads requires implementing and maintaining security controls that safeguard containers and their underlying infrastructure. Docker container security practices are designed to protect containerized applications from risks like security breaches, malware and bad actors.

Since Docker containers are isolated from each other and the host system, they have an inherent level of security by design. However, this isolation is not absolute. Docker security revolves around a holistic zero trust framework that encompasses the runtime, build and orchestration of containers.

The need for Docker and other container-related security has increased the popularity of DevSecOps. Which is a security approach that automates the integration of security practices at every phase of the software development lifecycle—spanning initial design through integration, testing, delivery and deployment. Also, Docker security best practices include third-party container security tools and solutions, including scanning and monitoring, that can detect security issues before they impact production.

## Docker use cases

**Cloud migration**
Docker's portability simplifies and speeds the cloud migration process from across diverse environments—whether moving data, applications and workloads from an on-premises data center to a cloud-based infrastructure or from one cloud environment to another.

**Microservices architecture (microservices)**
According to Statista, more than 85% of large global organizations use microservices for their application development[11]. Docker simplifies app deployment as each microservice can be containerized and independently scaled and managed, thus eliminating the need for developers to configure and manage particular environments.

**Continuous integration and continuous delivery (CI/CD)**
Docker is ideal for continuous integration and continuous delivery (CI/CD) pipelines as it provides a consistent environment for testing and deploying applications, reducing possible errors during deployment.

**DevOps**
The combination of microservices as a software development approach and Docker creates a solid foundation for DevOps teams. It allows them to adopt agile practices so they can iterate and experiment rapidly, which is crucial to delivering software and services at the speed the market demands.

**Hybrid multicloud deployment**
A lightweight containerized technology like Docker makes moving applications across different environments easy. All major cloud service providers offer Docker-related development and management services that support running Docker in hybrid cloud environments, which unify on-premises, public cloud, private cloud and edge settings. Docker can be easily deployed across multicloud IT infrastructure, which refers to cloud services from more than one cloud vendor.

**Containers as a service (CaaS)**
Containers as a service (CaaS) enables developers to manage and deploy containerized applications, making it easy to run Docker containers at scale. All the major CSPs offer CaaS as part of their cloud services portfolios, along with infrastructure as a service (IaaS), software as a service (SaaS), and so forth.

**Artificial intelligence/machine learning (AI/ML)**
Docker speeds artificial intelligence and machine learning development with fast, easy, portable application development, accelerating innovation and time to market. Also, Docker Hub is home to hundreds of AL/ML images that further support AL/ML development teams. In 2023, Docker launched Docker AI[12], which offers developers context-specific, automated guidance when they are editing a Dockerfile or Docker Compose file.