Docker Components and Use Cases

Chiradeep BasuMallick

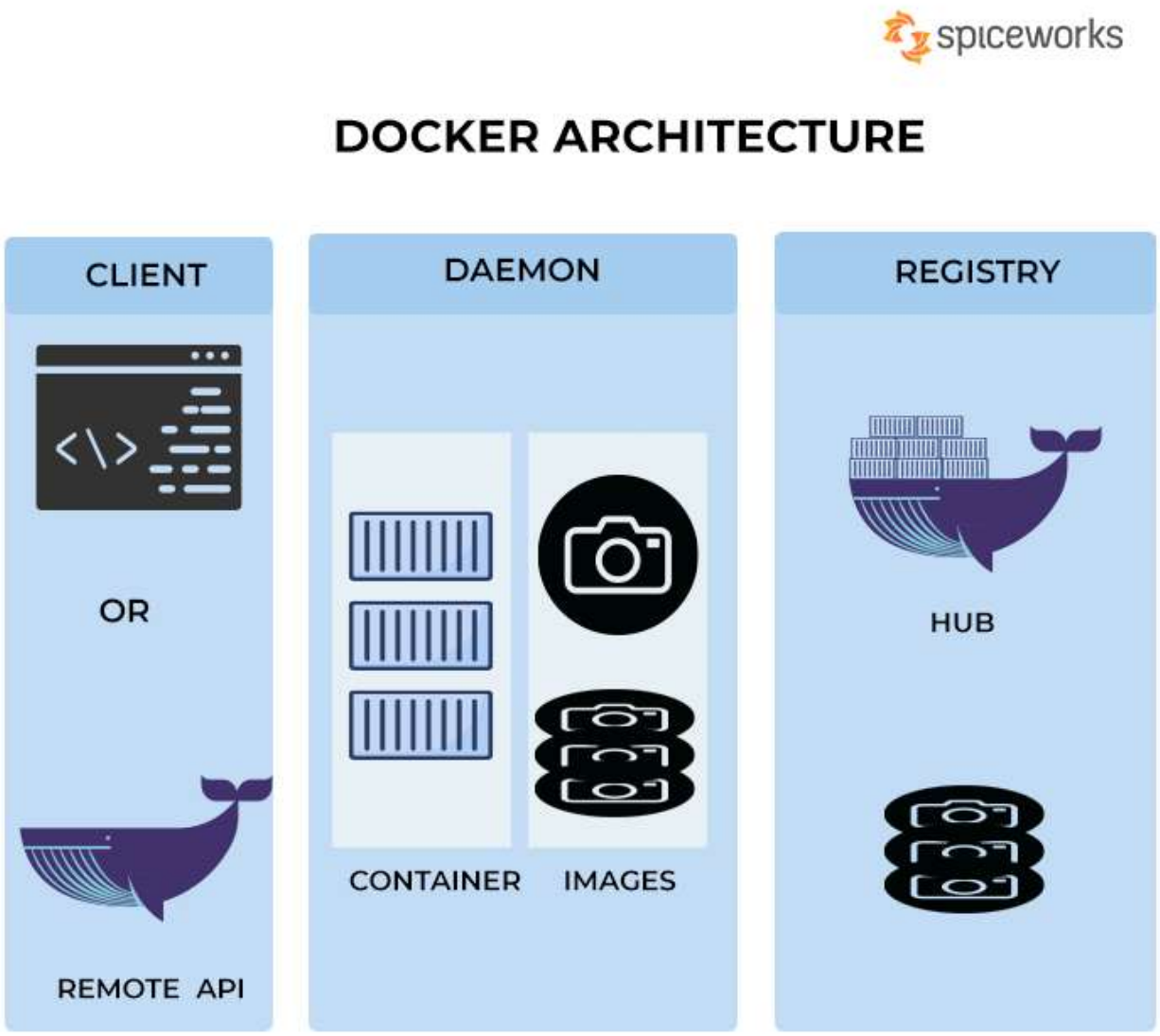
15 min read

https://www.spiceworks.com/tech/big-data/articles/what-is-docker/

Docker is a freemium platform as a service (PaaS) solution that aids in creating, operating, and maintaining containers for isolated software development and testing by creating a virtualized operating system (OS) environment. This article explains the working of Docker, its components, and top use cases.

What Is Docker?

Docker is defined as a freemium platform as a service (PaaS) solution that aids in creating, operating, and maintaining containers for isolated software development and testing by creating a virtualized operating system (OS) environment.



Docker Architecture

Rapid application development, testing, and deployment are made possible by the solution known as Docker. Docker packages software into standardized units called containers containing all the necessary code, libraries, system tools, and runtime. With Docker, you can quickly scale and deploy apps into any environment, confident that your code will work.

Five years ago, Solomon Hykes participated in founding the company Docker, which aimed to simplify the use of containers. The hype intensified with the June 2014 release of Docker 1.0. Distributed application development and delivery are made more accessible by containers. As businesses switch to cloud-native development and hybrid multi-cloud settings, they have grown in popularity.

Without Docker, developers can create containers by utilizing features already present in Linux and other OS systems. However, Docker accelerates, simplifies, and secures containerization. When you wanted to operate a web application in the past, you would purchase a server, install Linux, build up a LAMP stack, and then launch the application. To prevent the application from crashing due to excessive traffic, you should practice proper load balancing by creating a second server if your app becomes popular.

However, as technology has advanced, the internet is based on clusters of redundant, interconnected servers, referred to as “the cloud.” Docker uses this system to free application development for hardware infrastructure lock-in.

Over 37 billion containerized programs have been downloaded, claims Docker, and over 3.5 million applications have been packaged into containers. It has been adopted by practically all IT and cloud computing service providers. High-profile technology companies like Microsoft, IBM, and Red Hat have taken notice of it, and so have venture capitalists who are eager to invest millions of dollars in this innovative company.

See More: What Is a Data Catalog? Definition, Examples, and Best Practices

How Does Docker Work?

Docker employs a client-server design. A Docker client communicates with the daemon (that develops, operates, and distributes Docker containers). REST application programming interfaces (APIs) are used for network communication and UNIX sockets between the client and daemon.

The client may be connected to a daemon remotely, or the developer can run the daemon and client of Docker on the same computer. You may also interact with applications made up of a set of containers by using Docker Compose, which is another client for Docker.

The container's connection to the default network will be made possible through a network interface. After that, Docker will launch the container and run /bin/bash. You can type input while the result is logged to the terminal because the container is connected to the terminal and operating interactively. You can type exit to end the /bin/bash command, and the container will shut down.

Regarding container-based implementation, Docker has taken over as the de facto standard. Developers utilize Docker, an open-source platform created by DotCloud and implemented in the GO programming language, to create, distribute, and operate programs. With Docker, it is simple to segregate apps from the surrounding IT infrastructure for a project's quick delivery. It allows programmers to containerize and run the application in loosely remote environments.

Docker provides a level of separation that allows developers to run numerous containers concurrently on a single host. This allows for more flexibility. Originally based on Linux Containers, Docker currently uses the container runtime libcontainer function (a part of runc). The Linux kernel's built-in virtualization and process isolation features enable containers. Similar to how a hypervisor allows multiple virtual machines (VMs) to share the CPU, memory, and other resources of a single hardware server.

These capabilities, such as control groups (cGroups) for allocating resources among processes and namespaces for restricting a process's access or visibility into other resources or sections of the system, allow the different parts of an application to share the resources of a single instance of the host operating system.

Although Docker is not required to create containers, the platform's set of tools, dependencies, and libraries needed to run an application make container creation simpler, safer, and more manageable.

One can build modern platforms and apps with the help of Docker as a fundamental building piece. Undoubtedly, Docker makes constructing and running distributed microservices architecture, publishing code into the continuous integration and delivery (CI/CD) pipeline, and developing scalable data processing systems simple. A constant and quick distribution of an application is made possible by Docker. A developer's laptop, a cloud provider's virtual machine, a data center, or a mixed environment can execute Docker.

See More: What Is Data Governance? Definition, Importance, and Best Practices

8 Key Components of Docker

The software comprises various components, each contributing significantly to the platform. These are:

1. The Docker Engine

The Docker Engine, also known as simply DE, is the essential central component of the Docker system. It must be downloaded and installed on the host computer. Next, a lightweight runtime system and the client-server technology that lies on top of the DE are used in creating and managing containers. There are three parts to the Docker Engine:

Server: The Docker daemon (dockerd) is in charge of creating and managing containers.

Rest API: The Rest API enables the communication between applications and Docker and gives Dockerd instructions.

Command Line Interface (CLI): Docker commands are executed using the CLI.

2. Docker images

The building blocks for containers are called Docker images. Like snapshots for virtual machines, Docker images are immutable, read-only files that include the source code. It also contains libraries, tools, dependencies, and additional files to run an application.

Images are essential for reducing disk utilization, enhancing reusability, and accelerating Docker build. The importance of maintaining compact images cannot be overstated because you want your containers to be quick and light. A Dockerfile, which contains detailed instructions for constructing a specific Docker image, is used to create each image. You can develop images and customized containers more quickly and easily if you've mastered producing Docker images from Dockerfiles.

3. Dockerfile

A Dockerfile is a script with instructions on how to make a Docker image. In these instructions, you can find information about the operating system, languages, environment variables, file locations, network ports, and other details needed to run the image. The commands in the file are placed in groups, and those groups are automatically executed.

An image contains several layers. A new read-write layer is introduced once a Docker image has been launched to construct a container. The container layer is another name for this. With the additional layer, you can

modify the base image and then commit your modifications to produce a fresh Docker image for usage in the future.

4. The Docker Hub

Docker Hub is the most extensive cloud-based archive of Docker's container images. More than 100,000 images produced by open-source initiatives, software companies, and the Docker community are made accessible for use. The platform enables you to swiftly integrate your applications into a development pipeline, engage with team members, and ship your applications anywhere.

Like GitHub, developers can choose whether to maintain their private or public container images on Docker Hub by pushing and pulling them. Users of Docker Hub can freely distribute their images. To use the Docker filesystem as a starting point for any containerization project, they can also download prepared base images from it.

See More: What Is Enterprise Data Management (EDM)? Definition, Importance, and Best Practices

5. Docker volumes

When preserving data generated by a container that is already operating, using Docker volumes is a superior choice to adding extra layers to an image. With this feature's assistance, users can store data, transfer it across containers, and mount it to new ones.

Because they are kept on the host, Docker volumes are immune to any changes that may occur throughout the container's life cycle. When starting a container, you may generate and mount a Docker volume in several ways. These methods are all available.

6. Docker Compose

The process of creating and testing multi-container applications is made more accessible with the help of Docker Compose. It chooses the services to include in the application and generates a YAML Ain't Markup Language (YAML) file. One can use a single command to deploy and run containers. Docker Compose is a helpful tool to streamline the process of running and managing numerous containers simultaneously.

It connects the containers that must cooperate and manages them with a single coordinated command. You can also define nodes, storage, and service dependencies using Docker Compose. Other technologies like Kubernetes and Docker Swarm allow more complex variations of similar tasks, known as container orchestration.

7. Docker Desktop

Docker Desktop was known initially as Docker for Windows and Docker for Mac. When using Docker Desktop, it takes a few minutes to start creating and operating containers on either Windows or Mac. The entire Docker development environment may be installed and set up quickly.

The environment may include components like Kubernetes and Credential Helper, and Docker features such as the DE, Docker Compose, Docker CLI client, and Docker Content Trust. On any cloud platform, the tool is used to create and share containerized apps and microservices in various languages and frameworks.

8. Docker containers

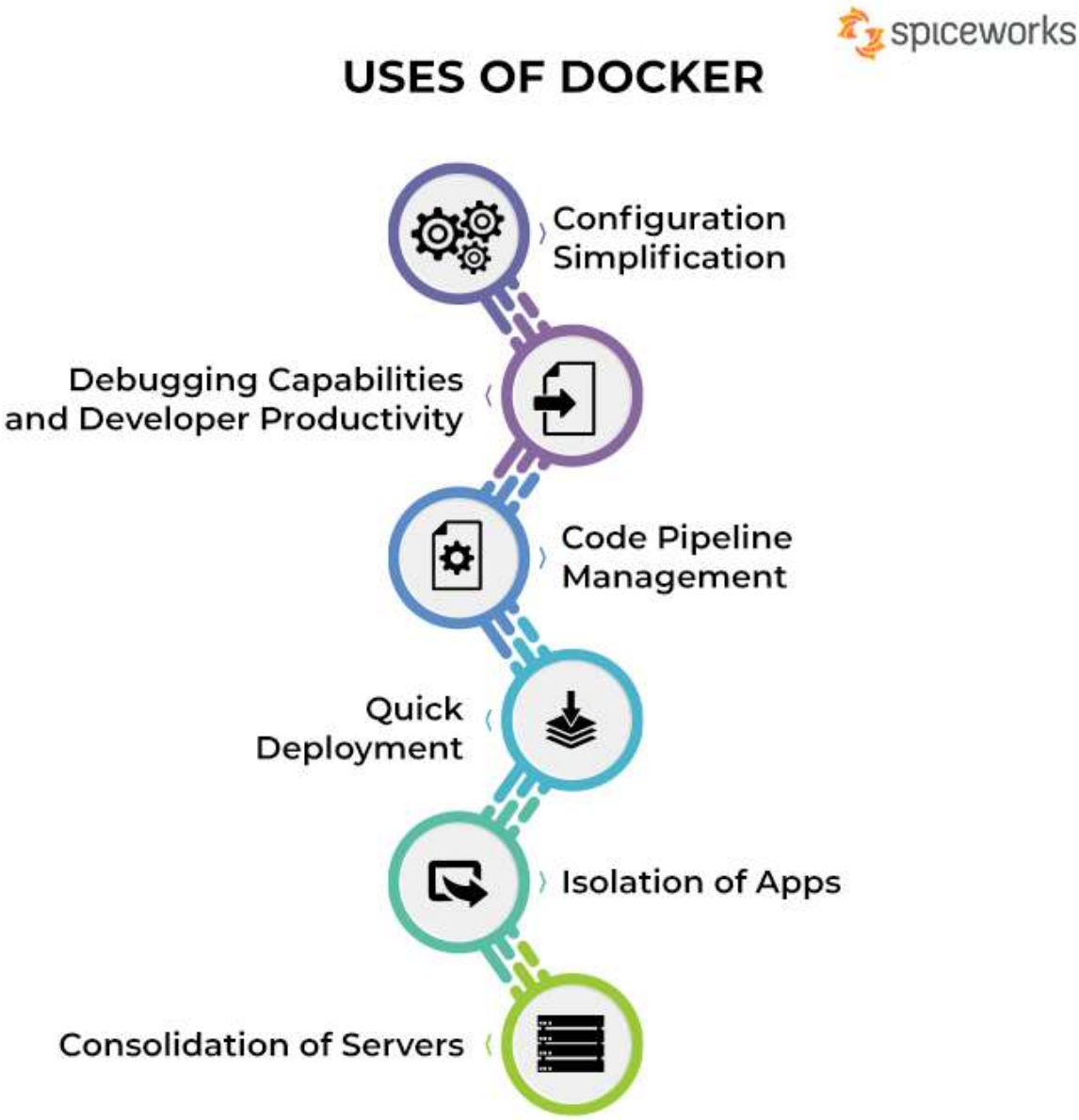
The active, operational instances of Docker images are known as containers. Docker images are read-only files, whereas containers contain executable, transient content. Users can interact with them, and administrators can use Docker commands to change their parameters.

You can regulate the container storage's isolation, computer network, and other fundamental subsystems from one another and the host machine. Developers will utilize any configuration settings given to a container when it is being created or started to define the container and the image it contains. When a container is deleted, any state changes made but not preserved in persistent storage are lost.

See More: Top 8 Big Data Security Best Practices for 2021

8 Essential Uses of Docker

While Docker is primarily used in containerization scenarios, it has many other use cases and applications. DevOps teams, for example, regularly use the solution to speed up software delivery. Other important uses include:



Uses of Docker

1. Configuration simplification

Docker's primary use case is configuration simplification. One of the most significant advantages of virtual machines is that they allow you to run applications with unique configurations on top of your existing

infrastructure. The overhead of a virtual computer is eliminated with Docker, which offers the same functionality.

It enables you to deploy code that incorporates your environment and configuration. Different environments can all utilize the same Docker configuration. This separates the needs of the application environment from those of the infrastructure. Docker may provide the flexibility to run your applications across several infrastructure as a service (IaaS) or platform as a service (PaaS) environments without requiring further adjustments. This is the ultimate aim that Docker may help you achieve. Docker is now supported by all IaaS and PaaS providers, including Google and Amazon.

2. Debugging capabilities and developer productivity

It is simpler for developers to investigate errors since containers enable them to replicate the application environment on their systems (or testing environments, development clusters, etc.). One of Docker's most valuable features is the ability to save a container's state to an image and execute it with another runtime setting.

It is also a huge time saver to quickly enter a running container and use development or debugging tools without worrying about ruining a typical, difficult-to-restore application security testing environment. One can promptly delete this modified and useless container and restore the original version.

3. Code pipeline management

There are several environments that the code must pass through while moving from developer machines to production. Minor variations between any of them are possible along the process. Docker can streamline the process of building and distributing code. This is because it provides a

stable application environment at every process stage, from development to production.

Also, due to the immutable nature of Docker images and the simplicity with which one can use them, it guarantees that there will be no change in the application runtime environment from development to production.

Rapid deployments that result in shorter feedback loops are necessary for innovation. Application turnaround is accelerated by Docker's local distribution of images and caching layers, and task execution is made possible by containers' lightning-fast startup rates.

4. Quick deployment

It took days to set up a new hardware resource before virtual machines. This amount was reduced to minutes by virtualization. Docker decreases the time to mere seconds by constructing containers meant for processing without launching an operating system. This is the underlying technology that allowed Google and Facebook to use containers.

In essence, you do not have to bother about the expense of resource management in your data center when you build or remove them. By utilizing a more aggressive resource allocation, it is simple to increase the usual data center utilization rate, which is much lower. Additionally, a more aggressive resource allocation is possible due to how inexpensive it is to start a new instance.

5. Isolation of apps

You may want to run many isolated programs on the same machine for various reasons. The productivity flow for developers mentioned previously is an illustration of this. However, there are also other instances. Server consolidation to cut costs or a progressive plan to break

up a monolithic application into separate parts are two examples of such situations.

Let us consider the scenario where a developer has to manage two flask-based REST API servers. However, each of them makes use of a slightly different flask version and other dependencies. Running these API servers under several containers offers a simple solution to dependency complexities.

6. Consolidation of servers

The application isolation capabilities of Docker allow merging several servers to reduce costs, much like how VMs are used to combine multiple apps. Docker enables a considerably denser server consolidation than virtual machines since it does not carry the memory storage cache of numerous operating systems and can disperse unused memory between instances. The new generation of highly customized PaaS, such as Elastic Beanstalk, Heroku, and App Engine, uses Docker's strong container capacity.

In addition, open-source projects like Deis, Kubernetes, cAdvisor, and Panamax make it feasible to manage the distribution and management of a large number of containers that represent a multi-tier application architecture.

7. Application security

Docker containers offer a more secure environment for application workloads than conventional server and virtual machine (VM) models. They allow you to divide your applications into smaller, loosely coupled parts that are all separate from one another and have a much smaller attack surface.

Containers increase the implicit separation of system parts, enabling a better security level by default. However, because user ID isolation is still lacking, running apps with various security profiles on the same host is still dangerous. Docker allows the usage of a non-default profile and permits the addition and removal of capabilities. This could make Docker less secure by removing capabilities or more secure by adding capabilities, depending on the application security engineer's skill level.

8. Multi-tenancy

The usage of Docker in multi-tenant systems, which prevents the need for extensive application rewrites, is yet another intriguing use case. An instance is multi-tenancy's rapid and straightforward development for an internet of things (IoT) application. Such multi-tenant systems have much more intricate, rigid, and challenging to manage code bases. Rearchitecting an application takes a lot of effort, and it is expensive.

It is simple and affordable to build segregated environments using Docker for running many app layer instances for each tenant. Due to Docker environments' quick startup times and simple API, which you can utilize to spin containers dynamically, it is possible for all development tasks.

See More: What Is Deepfake? Meaning, Types of Frauds, Examples, and Prevention Best Practices for 2022

Takeaway

Docker has consistently ranked among the top solutions for creating containers, which is more important than ever in the era of hybrid working. The *2020 Stack Overflow Developer Survey* found that Docker was the No.1 platform most developers wanted to use and the No.2 most loved. As remote and hybrid work continues to be the new normal, infrastructure managers, developers, and DevOps engineers need new digital tools to

do their jobs efficiently. Docker is an essential part of this toolkit and central to your IT resilience and agility.

*Did this article help you understand how Docker works and its main use cases? Tell us on Facebook*Opens a new window *, Twitter*Opens a new window *, and LinkedIn*Opens a new window *. We'd love to hear from you!*

MORE ON DATA

- What Is Data Fabric? Definition, Architecture, and Best Practices
- Why the Future of Database Management Lies In Open Source
- What Is Data Security? Definition, Planning, Policy, and Best Practices
- Top 10 Data Governance Tools for 2021
- How Synthetic Data Can Disrupt Machine Learning at Scale