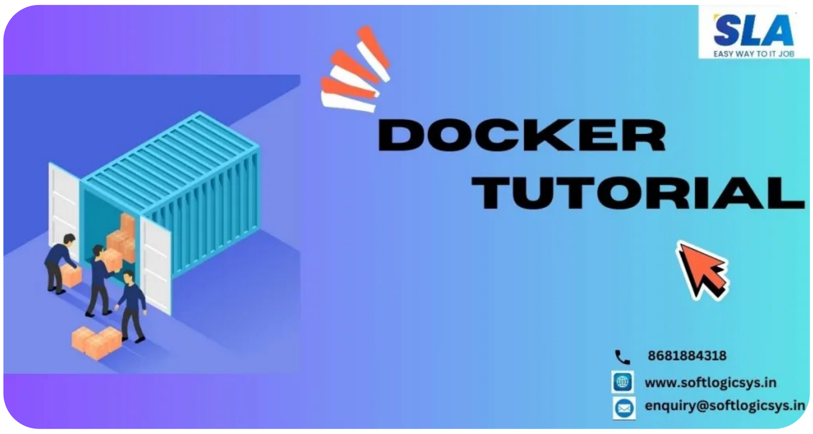

Share on your Social Media

Docker Tutorial

Published On: September 5, 2024

Docker is a suite of platform-as-a-service offerings that provide software in packages known as containers by utilizing OS-level virtualization. Learn everything you need to know in this Docker tutorial and become an expert DevOps engineer with Docker skills.

Download Docker Tutorial PDF

Introduction to Docker

A software platform called Docker makes it easier to build, test, and implement programs quickly. Docker software bundles programs into uniform units known as containers, each of which contains the libraries, runtime, code, and system tools required for the program to function. Let's explore the following in this Docker tutorial:

- Overview of Docker

Related Courses at SLA

- Docker Online Training
- Docker Training in OMR
- Docker Training in Chennai

Related Posts

- Docker Setup and Configuration
- Components of Docker
- Advanced Docker Components
- Docker Commands
- Docker Networking
- Docker Registry

Overview of Docker

Developers utilize Docker, an open-source platform, to automate the deployment of applications inside containers. Docker provides a uniform environment that allows the software to execute in a variety of computing settings. Docker makes it easy to ship, build, and execute applications more efficiently.

Docker offers a platform and extensive functionality for controlling the lifetime of containers.

- Containers can be used to construct applications and sustain their parts.
- For all of your apps, containers can serve as the distribution and testing unit.
- Using Docker, you can easily and consistently deploy apps across all environments, including hybrid infrastructures, cloud platforms, and local data centers.

Advantages of Docker

The advantages of Docker:

- Docker allows for quick deployment.
- Unlike typical virtual machine settings, the environment itself is highly portable and was created with efficiency.
- It enables you to operate several Docker containers in a single environment.
- The Docker environment you wish to establish can be described using a language called YAML, which can be used to automate the configuration itself. You can then swiftly scale your environment as a result.
- Security is arguably the most important benefit

these days. Docker is highly concerned about security.
- It is one of the essential elements of the system’s agile architecture.

VBA Macros Tutorial for Beginners VBA macros are programs that automate repetitive operations in Microsoft...

Docker Salary

Docker Setup and Configuration

Docker has become more stable, making it simple to set up and operate on your preferred operating system. Docker has recently made considerable investments to enhance its users’ onboarding experience on Windows and Linux operating systems, making Docker today very easy to run.

Setting up Docker on Mac, Linux, and Windows in the given links.

After installing Docker, use the following commands to verify that it is installed correctly:

```
$ docker run hello-world

Hello from Docker.
This message shows that your installation appears to be working correctly.
...
```

Components of Docker

In this Docker tutorial, we will go over four components:

- Docker client and server
- Docker image
- Docker registry
- Docker container

Docker Client and Server

This is a solution that requires you to utilize the terminal on your Mac or Linux computer to send commands to the Docker daemon from the Docker client.

- A REST API is used to facilitate communication between the Docker host and the client.
- Docker Pull command, which instructs the

daemon to carry out the task via interacting with other components (image, container, registry).
- The Docker daemon functions as a server that communicates with the operating system and provides services.
- The Docker daemon continuously scans the REST API for requests to see if it needs to execute any particular ones.
- Using the Docker command within your Docker daemon will initiate all of your performances if you wish to initiate and start the entire procedure.
- The Docker daemon and registry can then be executed on a Docker host.

Docker Image

A template with instructions for the Docker container is called a Docker image. YAML, or Yet Another Markup Language, is the language used to write the template.

- The YAML file contains the Docker image, which is then hosted as a file on the Docker registry.
- There are multiple important layers in the image, and each layer is dependent on the layers below it.
- Each command in the Dockerfile is executed to produce an image layer, which is in read-only format.
- There is a layer of dependencies above your base layer, which is normally where your base image and base operating system are located.
- These then make up the read-only file that would eventually form your Dockerfile, including the instructions.

Understanding the distinction between base and child images is crucial when it comes to photographs.

- **Base images** are those without a parent

image; they typically contain an operating system such as Ubuntu, BusyBox, or Debian.
- **Child Images:** Pictures that extend the capability of basic pictures are called child images.

Next, there are user and official photos, which can be base or child images.

Official Images: Images that are officially supported and maintained by the Docker team are known as official images. Usually, these are one word in length.

The pictures of Busybox, Ubuntu, Python, and Hello World in the above list are official images.

User Images: Images made and shared by users, just like you and me, are known as user images. They enhance base images with new capabilities. These are commonly formatted as user/image-name.

Docker Registry

The Docker registry is the location from which you would distribute and host different image formats. The repository is nothing more than an assembly of Docker images, which are created using YAML instructions and are incredibly simple to share and save.

- The Docker images can be named and tagged to make it simple for users to locate and distribute them inside the Docker registry.
- Using the publicly accessible Docker Hub registry, which is open to everyone, is one method to begin administering a registry.
- For internal usage only, you can also establish your register.

You can build both public and private photos for the registry that you internally establish.

Push and pull are the commands you would use to link the registry.

- To push a newly generated container environment from your local management node to the Docker registry, use the push command.
- To receive newly produced clients (Docker images) from the Docker registry, use the pull command.

Once more, a Pull command pulls and retrieves a Docker image from the Docker registry, while a Push command enables you to push a newly produced command to the registry, whether it be your private registry or Docker hub.

Docker Syllabus PDF

Docker Container

The Docker container contains all the instructions needed to run the solution you've selected as an executable package, together with all of its dependencies. Because there is inherent structural redundancy, it is incredibly lightweight.

- The container is naturally portable. Its full isolation while operating is an additional advantage.
- Unlike with a virtual machine or non-containerized environment, it is guaranteed not to be affected by any host OS security measures or special configurations, even if you are running a container.
- One extremely helpful feature of Docker environments is their memory sharing across several containers.
- This is especially true if you have a virtual machine with a fixed amount of memory allocated to each environment.

Take a look at this simple Docker run command example, which launches a single Redis container:

$ Docker run redis

It will be taken from the registry if the Redis image is not installed locally. Following this, you will be able to use the newly created Docker container Redis in your environment.

Advanced Docker Components

Following an overview of Docker's many components, this lesson will now concentrate on its more sophisticated components:

- Docker Compose
- Docker Swamp

Docker Compose

Several containers can be run as a single service with Docker-compose. It accomplishes this by enabling communication across the containers while maintaining isolation for each one. As previously said, YAML would be used to write the compose environments.

Example: It would be if you had a single database running on an Apache server and you needed to launch more containers to execute more services without starting them one at a time. To do that, you would use Docker compose to write a group of files.

Docker Interview Questions and Answers

Docker Swamp

Within the Docker platform, developers and IT managers can establish and oversee a cluster of swarm nodes using the Docker Swarm service for containers.

- A Docker daemon is a node in the Docker Swarm that communicates with other Docker daemons using the Docker API.
- Manager nodes and worker nodes are the two different kinds of nodes that form a swarm.
- Cluster management tasks are maintained by

a manager node.
- The manager node sends tasks to worker nodes, which then completes them.

Docker Commands

An open-source project called Docker automates the deployment of programs as portable, stand-alone containers that can operate on-site or in the cloud.

With Docker, you can segregate your apps from your infrastructure, facilitating rapid software delivery and enabling you to manage your infrastructure in the same manner as your applications.

Docker Run Command

Using this command, a container can be launched from an image. The docker creates and docker start commands are combined to form the docker run command.

$ docker run <image_name>

$ docker run –name <container_name> <image_name>

Docker Pull Command

You can use this command to pull any image that is available in Docker Hub, the official registry for Docker. It pulls the most recent image by default, but you can specify the image's version.

$ docker pull <image_name>

Docker PS Command

This command displays a list of all the containers that are currently operating. With it, we may use different flags.

-a flag: Indicates whether a container is running or stopped.

-l flag: It displays the most recent container.

-q flag: It just displays the container's ID.

$ docker ps [options..]

Docker Stop Command

If a container has crashed or you wish to switch to another, you can use this command to stop it.

$ docker stop <container_ID>

Docker Start Command

In the unlikely event that you need to restart the stopped container, this command can assist you in doing so.

$ docker start <container_ID>

Docker RM Command

It is used to remove a container. When a container is created, it automatically receives an ID and a made-up name, such as confident_boyd, heuristic_villani, etc. Either the ID or the name of the container may be mentioned.

-f flag: remove the container with force.

-v flag: the volumes are removed.

-l flag: eliminate the specified link.

$ docker rm {options} <container_name or ID>

Docker RMI Command

This is used to remove the picture from Docker. To clear up space, you can remove any unnecessary images from the Docker local storage.

docker rmi <image ID/ image name>

Docker Images

This is a list of every pulled image that is in our system.

$ docker images

Docker EXEC Command

We can execute new instructions in an active container by using this command. This command does not restart when the container restarts; it is only effective while the container is operating.

-d flag: To run the tasks in the background, use the -d parameter.

-i flag: even when not attached, it will maintain STDIN open.

-e flag: It sets the environment variables with the -e flag.

$ docker exec {options}

Docker Port Mapping: Docker Ports

We must map the port on our host—our laptop, for example—to the port on the container to access the Docker container from the outside world. Port mapping is used in this situation.

$ docker run -d -p <port_on_host>

<port_on_container> Container_name

Docker Login

You may push and pull your images by using the Docker login command to help you authenticate with the Docker hub.

docker login

After providing your login and password, DockerHub will authenticate you and allow you to carry out the tasks.

Docker Push

Following the creation of your customized image using Dockerfile, you must push your image using the following command to save it in the remote registry, DockerHub.

docker push <Image name/Image ID>

Docker Build

Dockerfile is used in conjunction with the docker build command to create the docker images.

docker build -t image_name:tag.

Use the name of the image you built with and enter the tag number instead of image_name. The current directory is represented by a “dot.”

Docker Stop

Docker containers allow you to start and stop them so that you can perform container maintenance. You can use the following commands to start and stop certain containers.

docker stop container_name_or_id

Stop Multiple Containers

Rather than halting just one container. With these commands, you can stop more than one container at once.

docker stop container1 container2 container3

Docker Restart

You can run into issues and have the containers fail to start when using Docker to execute the containers. Using the following instructions, you can restart the containers to resolve the containers.

docker restart container_name_or_id

Docker Inspection

Real-time issues may arise with Docker containers. To troubleshoot these failures, use the following instructions.

docker inspect container_name_or_id

Docker Commit Command

Once the containers have been run using the current image, you can update them by interacting with them. From there, you can use the following commands to produce an image of the containers.

docker commit container_name_or_id new_image_name:tag

Docker Networking

With Docker Networking, you may establish a network of Docker containers under the management of a manager master node.

- A network is a set of two or more devices that can interact virtually or physically.
- Docker constructed a virtual network called the Docker network to facilitate communication between Docker containers.
- Ports do not need to be open to the host computer for two containers operating on the same host to communicate.
- Whether your Docker hosts are running Linux, Windows, or both, you may use Docker to manage them in any platform-specific way.

Docker Training

Network Drivers

Docker comes with many default network drivers that can be installed with the aid of plugins. The command to view the Docker list of containers is provided below.

docker network ls

Types of Network Drivers

bridge: If you construct a container without indicating the type of driver, it will only be built in the default network, the bridge network.

host: Since containers are formed directly in the system network, they will not have an IP address, eliminating any separation between the host and containers.

None: Containers won't be given IP addresses. We cannot enter these containments from the outside or any other container.

Overlay: An overlay network will facilitate communication between various Docker swarm services and allow connections between several Docker demons.

IPvlan: By using the IPvlan driver, users have total control over IPv4 and IPv6 addressing.

macvlan: A container's MAC addresses can be assigned thanks to the macvlan driver.

Use the Default Network to Launch a Container

1. Understanding the Docker Network Command

Your Docker Network can be created, managed, and configured primarily with the help of the Docker Network command.

Let's examine the possible uses for the Docker Network command's sub-commands. to gain additional knowledge about Setting Up a Network in Docker and Linking a Container to It.

sudo docker network

2. Making Use of Docker Network Establish command

We can establish our own Docker network and deploy our containers within it by using the "create" command.

sudo docker network create –driver <driver-name> <bridge-name>

3. Making use of the command Docker Network Connect

You can link an active Docker container to an already-existing network by using the “Connect” command.

sudo docker network connect <network-name> <container-name or id>

4. Making use of the command Docker Network Inspect

A Docker network’s details can be obtained by using the Network Inspect command.

sudo docker network inspect <network-name>

5. Using the ls command on the Docker Network

The list command can be used to get a list of every Docker network.

sudo docker network ls

6. Making use of the command Docker Network Disconnect

To remove a container from the network, use the disconnect command.

sudo docker network disconnect <network-name> <container-name>

7. Using the rm command in Docker Network

With the rm command, a Docker network can be deleted.

sudo docker network rm <network-name>

Here, you must ensure that no container is referencing the network at the moment if you wish to remove it.

8. Applying the prune command on the Docker Network

You can use the prune command to get rid of all the unnecessary Docker networks.

sudo docker network prune

Common Operations

- **docker network inspects:** Using the "docker network inspect" command, we can look at the configuration details of a particular network, including its name, the containers that have been linked to it, the kind of driver that was used to build it, and other details.
- **docker network ls:** Using "docker network ls," we can view every network that is accessible on the current host.
- **docker network creates:** We can construct a new network by using the command "docker network create" along with the driver name (bridge, overlay, or macvlan).
- **docker network connects:** We need to make sure the right network has already formed on the host before we can use this command. The container can then be connected to the required network by using Docker "network connect."

Docker Registry

The distributed and centralized Docker Registry system is used to gather and manage Docker images.

- It offers both private and public repositories based on the user's preference for whether or not to make the image publicly accessible.
- It is a crucial part of the workflow for containerization since it makes application deployment and administration more efficient.

What is Docker Registry?

A mechanism for distributing and keeping track of Docker images with unique names is called a registry. The same image may exist in multiple variants, each with a unique collection of tags.

- Each change to an image is kept in a different Docker repository, which is a component of a Docker registry.
- Docker users can use the registry to add new images and retrieve images locally if the necessary access permissions are given.
- To store and distribute Docker images, a server-side application known as the registry is utilized. It is stateless and highly scalable.

Different Types of Docker Registries

The numerous types of Docker registries are as follows:

- DockerHub
- Amazon Elastic Container Registry (ECR)
- Google Container Registry (GCR)
- Azure Container Registry (ACR)

Docker Hub

Users can save and distribute Docker images via DockerHub, a cloud-based repository offered by Docker. It serves as a central location for developers to locate ready-made images for a range of software programs.

By the decision of whether or not to disclose the Docker images publicly, it offers both public and private repositories for maintenance.

Amazon Elastic Container Registry (ECR)

One component of Amazon Web Services' cloud computing platform, Amazon Elastic Compute Cloud, lets users rent virtual machines to run their software.

Google Container Registry (GCR)

A private Docker repository that integrates with well-known continuous delivery platforms is called Container Registry. It is powered by the

Andromeda-based network fabric of Google Cloud, which offers reliable uptime on an infrastructure shielded by Google's security.

Azure Container Registry (ACR)

In addition to managing associated content formats like Helm charts, OCI artifacts, and images created by the OCI image format definition, Azure Container Registry also manages private Docker container images.

Basic commands for Docker registry

The fundamental commands for the Docker registry are as follows:

1. Start registry

Using port 5000, this operation essentially launches a Docker registry on your local computer or server.

docker run -d -p 5000:5000 –restart=always –name registry registry:2

It gives Docker instructions to start a registry with the name registry:2 in detached mode. If the registry fails, restart it right away by mapping port 5000 to a local port.

2. Pulling some images from the hub

The command to extract the image from the public Docker registry is as follows; in this case, the Ubuntu image is being pulled.

docker pull ubuntu:latest

3. Linking to your registry and tagging that picture

The command to tag the image and point to your registry is as follows:

docker image tag ubuntu:latest localhost:5000/gfg-image

4. Pushing the image

To push the image from Dockerhub, use the following command:

docker push localhost:5000/gfg-image

5. Pulling that image back

With this command, Docker is told to retrieve the gfg-image image from the local registry, which is running on port 5000 on localhost.

docker pull localhost:5000/gfg-image

6. Stop the registry

Docker is told to shut the registry container using this command:

docker container stop registry

7. Stop the registry and remove the data.

The command to successfully stop the registry and delete the related data is as follows:

docker container stop registry && docker container rm -v registry

How does the Docker Registry work?

- A framework for distributing and storing Docker images is offered by Docker Registry.
- Users have the option to submit and tag their Docker images with a name and version number in the registry.
- Subsequently, these photographs can be found and downloaded by other users from the registry.
- Docker Registry is available as a cloud-based service or can be self-hosted.
- Administrators may ensure that only authorized users and systems are accessible by implementing access controls and permissions with the help of the Docker Registry.

- Pipelines for continuous integration and continuous deployment, or CI/CD, are made easier by the Docker Registry.

Why is the Docker Registry important?

- Docker Registry makes it simple to share and distribute Docker images.
- It can save time and costs by making the management and deployment of Docker containers simpler.
- A vital part of the Docker ecosystem, the Docker Registry is utilized extensively by developers and businesses of all kinds.
    - **Version Control and History:** Docker Registry makes it easier to version Docker images, allowing developers to keep track of changes and roll back to earlier iterations as necessary.
    - **Collaboration and Community:** The developer community is encouraged to innovate, software development cycles are sped up, and knowledge sharing is facilitated by the Docker Registry.

Important Docker Registry Features

The main characteristics of the Docker Registry are as follows:

- **Docker Images Storage:** It may be efficiently stored and managed in a central repository with the help of centralized image storage.
- **Access Control:** It involves securing access to pictures by offering robust authorization and authentication procedures.
- **Integration with CI/CD:** It makes it easier to integrate deployment pipelines for automated processes and continuous integration.
- **Version Control:** It makes it simple to track and undo changes to an image by supporting versioning and labeling of the image.

Common Docker Registry Use Cases

The common use cases for Docker Registry are as follows:

- **Keeping and Sharing Images:** It facilitates the centralization of a repository service for the internal sharing and archiving of Docker images.
- **CI/CD Pipeline:** The smooth integration of Docker images into CI/CD pipelines for automated builds and deployments is made possible by continuous integration/continuous deployment, or CI/CD.
- **Version Control:** Version control ensures consistency and traceability between development and production environments by helping to manage the many Docker image versions.
- **Access Control and Security:** It ensures that only authorized users can push or pull images by offering secure access management and permissions for Docker images.

Conclusion

Docker streamlines the development lifecycle by allowing developers to work in standard environments with local containers hosting your applications and services. Learn Docker skills in our **Docker course in Chennai.**

Softlogic Systems

KK Nagar [Corporate Office]

No.10, PT Rajan Salai, K.K. Nagar, Chennai – 600 078.
Landmark: Karnataka Bank Building
Phone: +91 86818 84318
Email: enquiry@softlogicsys.in
Map: Google Maps Link

OMR

No. E1-A10, RTS Food Street
92, Rajiv Gandhi Salai (OMR),
Navalur, Chennai - 600 130.
Landmark: Adj. to AGS Cinemas
Phone: +91 89256 88858
Email: info@softlogicsys.in
Map: Google Maps Link

Corporate Training
Hire With Us
Job Seekers
SLA's Recently Placed Students
Reviews
Sitemap

Important Links

Disclaimer
Privacy Policy
Terms and Conditions

Courses

Python
Software Testing
Full Stack Developer
Java
Power BI
Clinical SAS
Data Science
Embedded
Cloud Computing
Hardware and Networking
VBA Macros
Mobile App Development
DevOps

Social Media Links

Review Sources

Google
Trustpilot
Glassdoor
Mouthshut
Sulekha
Justdial
Ambitionbox
Indeed
Software Suggest
Sitejabber