Ref link: <https://www.geeksforgeeks.org/devops-roadmap/>

Ref2: <https://github.com/iam-veeramalla/Docker-Zero-to-Hero?tab=readme-ov-file> // telugu guy

Ref3: <https://docs.docker.com/storage/volumes/>

**Table of content**

authenticate and push the docker images to ECR.

able to dockerize an application.

1. containers

understanding containers

docker image & containers

2. Image

docker image

managing docker images

docker pull

inspecting images

3. docker architecture

docker architecture

docker daemon

4. volumes

volumes

bind mount volumes

bind mount volumes in action

using volumes for local dev

docker volumes

TMPFS mount

5. building docker image

Dockerfile

Creating Dockerfile

Building DockerImages

Running a container from Custom Image

Investigate Container file system

Buliding ExpressJS API

Dockerfile and Building image for user

Running Container for user

Exploring Dockerfiles

Dockerfile reference

6. image tagging and versioning

Pulling Images using a Specifc Tag

Creating tags

Creating Version 2 of the Dashboard

Never Run Latest In Production

Image Variants

7. docker registries

Docker Registries

Docker Login

Docker push

8. debugging

Docker inpect

Logs

Running commands in containers

9. networking

How to communicate between containers

Docker Network

MongoDB Container

MongoExpress

Understanding Container Communication

Another example

10. docker compose

Docker Compose cmd

Services

Docker Network

Docker Compose Up

Exploring docker compose commands

Docker Volume

Docker Compose Documentation

Security and static analysis tools

Docker scan

Trivy

Distroless Images

Security Best Practices

**Table of Content**

* [What is Docker?](https://www.geeksforgeeks.org/introduction-to-docker/?ref=lbp#what-is-docker)
* [Why Docker is popular?](https://www.geeksforgeeks.org/introduction-to-docker/?ref=lbp#why-docker-is-popular)
* [What is a Dockerfile?](https://www.geeksforgeeks.org/introduction-to-docker/?ref=lbp#what-is-a-dockerfile)
* [What is Docker Architecture and How Docker Works?](https://www.geeksforgeeks.org/introduction-to-docker/?ref=lbp#what-is-docker-architecture-and-how-docker-works)
* [What is Docker Image?](https://www.geeksforgeeks.org/introduction-to-docker/?ref=lbp#what-is-docker-image)
* [What is Docker Container?](https://www.geeksforgeeks.org/introduction-to-docker/?ref=lbp#what-is-docker-container)
* [What is Docker Hub?](https://www.geeksforgeeks.org/introduction-to-docker/?ref=lbp#what-is-docker-hub)
* [What is Docker Compose?](https://www.geeksforgeeks.org/introduction-to-docker/?ref=lbp#what-is-docker-compose)
* [How to Download Docker Desktop?](https://www.geeksforgeeks.org/introduction-to-docker/?ref=lbp#how-to-download-docker-desktop)
* [Docker Commands](https://www.geeksforgeeks.org/introduction-to-docker/?ref=lbp#docker-commands)
* [Docker Engine](https://www.geeksforgeeks.org/introduction-to-docker/?ref=lbp#dockerengine)
* [Why to use Docker?](https://www.geeksforgeeks.org/introduction-to-docker/?ref=lbp#why-to-use-docker)
* [What is Docker For AWS?](https://www.geeksforgeeks.org/introduction-to-docker/?ref=lbp#what-is-docker-for-aws)
* [Difference Between Docker Containers and Virtual Machines](https://www.geeksforgeeks.org/introduction-to-docker/?ref=lbp#difference-between-docker-containers-and-virtualmachines)
* [Install Docker On Ubuntu](https://www.geeksforgeeks.org/introduction-to-docker/?ref=lbp#install-docker-on-ubuntu)
* [Sample Example: Containerizing Application Using Docker](https://www.geeksforgeeks.org/introduction-to-docker/?ref=lbp#sample-example-containerizing-application-using-docker)
* [Sample Example to Push an image to Docker Hub](https://www.geeksforgeeks.org/introduction-to-docker/?ref=lbp#sample-example-to-push-an-image-to-dockerhub)
* [Fetch and run the image from Docker Hub](https://www.geeksforgeeks.org/introduction-to-docker/?ref=lbp#fetch-and-run-the-image-from-dockerhub)
* [Benefits of Docker](https://www.geeksforgeeks.org/introduction-to-docker/?ref=lbp#benefits-of-docker)
* [Alternatives of Docker](https://www.geeksforgeeks.org/introduction-to-docker/?ref=lbp#alternatives-of-docker)
* [Docker Security](https://www.geeksforgeeks.org/introduction-to-docker/?ref=lbp#docker-security)

## What is Docker?

[Docker](https://www.geeksforgeeks.org/introduction-to-docker/)is an open-source [containerization](https://www.geeksforgeeks.org/containerization-using-docker/) 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](https://www.geeksforgeeks.org/what-is-docker-image/) as a [docker container](https://www.geeksforgeeks.org/virtualisation-with-docker-containers/) in any machine where docker is installed without depending on the[operating system.](https://www.geeksforgeeks.org/what-is-an-operating-system/)

## **Why 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:**Through 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.

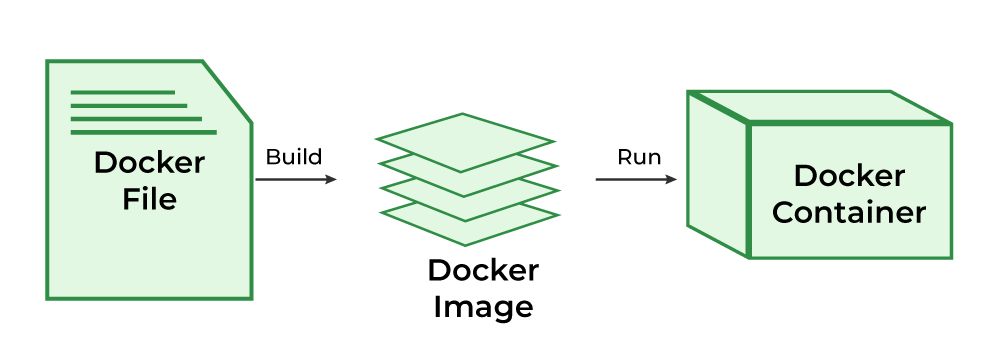
## **What is a Dockerfile?**

The [Dockerfile](https://www.geeksforgeeks.org/what-is-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 Dockerfile.



To Know more about the Dockerfile refer to the [Docker – Concept of Dockerfile](https://www.geeksforgeeks.org/docker-concept-of-dockerfile).

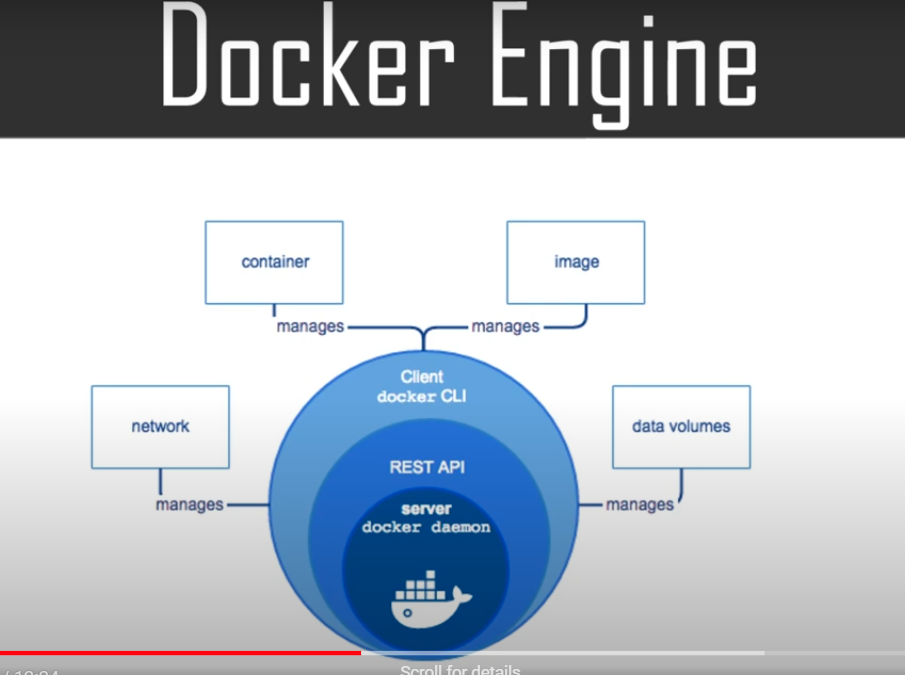
## **What is Docker Architecture and How Docker Works?**

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.

First understand Docker engine



**What is Docker Daemon?**

Docker daemon manages all the services

Like docker objects such as images, containers, networks, and volumes with the help of the API requests of Docker.

## **Docker Client**

The docker command uses the Docker API

The main objective of the docker client is to provide a way to direct the pull of images from the docker registry and run them on the docker host.

The common commands which are used by clients are **docker build, docker pull,**and **docker run.**

## **Docker Host**

that is responsible for running more than one container

**Docker Registry**

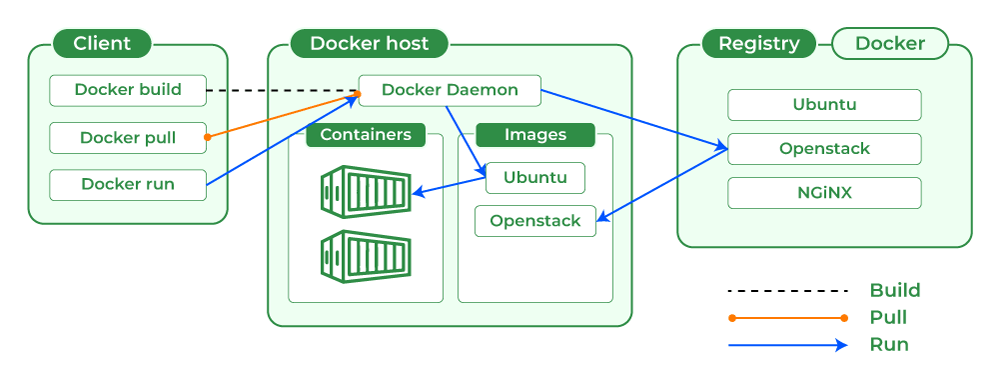
All the docker images are stored in the docker registry. There is a public registry which is known as a [**docker hub**](https://www.geeksforgeeks.org/what-is-docker-hub/)that can be used by anyone

## Docker images

An image contains instructions for creating a docker container. It is just a **read-only template**. It is used to store and ship applications.

## Docker Storage

[Storage driver](https://www.geeksforgeeks.org/data-storage-in-docker/)controls and manages the images and containers on our docker host.



## What is Docker Daemon?

Docker daemon manages all the services by communicating with other daemons. It manages docker objects such as images, containers, networks, and volumes with the help of the API requests of Docker

The main objective of the docker client is to provide a way to direct the pull of images from the docker registry and run them on the docker host. The common commands which are used by clients are **docker build, docker pull,**and **docker run**

**Understand docker architecture**

**See this youtube videos: https://www.youtube.com/watch?v=4Qv1tb1bm1Q**

## Docker Images

An image contains instructions for creating a docker container. It is just a **read-only template**. It is used to store and ship applications.

## **What is Docker Container?**

Containers are created from docker images as they are ready applications 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.

## What is 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.

## What is Docker Compose?

* **Docker Compose is a tool that lets you define multiple containers and their configurations via a YAML or JSON file.**

Docker Compose will execute a YAML-based multi-container application. The YAML file consists of all configurations needed to deploy containers [Docker Compose](https://www.geeksforgeeks.org/installation-of-docker-compose/), which is integrated with [Docker Swarm](https://www.geeksforgeeks.org/introduction-to-docker-swarm-mode/), and provides directions for building and deploying containers. With Docker Compose, each container is constructed to run on a single host.

**Docker life cycle:**

There are three important things,

1. docker build -> builds docker images from Dockerfile
2. docker run -> runs container from docker images
3. docker push -> push the container image to public/private regestries to share the docker images.



## Docker Commands

* **Docker Run:** It used to exeute commands during the build process of a docker image
* **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 container 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](https://www.geeksforgeeks.org/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**](https://www.geeksforgeeks.org/rest-api-introduction/)**:**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.

## Why to use Docker?

Docker can be used to pack the application and its dependencies which makes it lightweight and easy to ship the code faster with more reliability. Docker make its very simple to run the application in the production environment docker container can be platform independent if the docker engine is installed in the machine.

* **Resource Efficiency**: Docker helps in maximizing the resource utilization by running the multiple containers on a single host. It helps in reducing the infrastructure costs and improves the efficiency.
* **Version Control:** It simples the versioning for the applications and their dependencies ensuring the consistency and making easier of collaboration across the teams.
* **Microservices Agility:** It enables the adoption of microservices architecture, promoting the scalability, flexibility and fault isolation agile application development.

## What is Docker For AWS?

Docker is the most powerful tool to run the application in the form of containers. Docker container are light in weight and can be run on any operating system.

AWS provides the Amazon Elastic Container Service ([Amazon ECS](https://www.geeksforgeeks.org/introduction-to-amazon-elastic-container-service-ecs/)) it is an fully managed container service by which you can deploy, scale and manage the docker containers. Amazon ECS is the most reliable platform according to the performance and also it can be integrated with the other AWS Service like load balancing, service discovery, and container health monitoring. To know more about [Amazon Elastic Container Service (Amazon ECS)](https://www.geeksforgeeks.org/introduction-to-amazon-elastic-container-service-ecs).

## 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. |

## **Install Docker On Ubuntu // not read**

The following steps guide you in installation of docker on ubuntu:

### Step 1: Remove old version of Docker

* Execute the following command to remove the old versioned docker software:

$ sudo apt-get remove docker docker-engine docker.io containerd runc

### Step 2: Installing Docker Engine

* The following command is used for installation of docker engine:

$ sudo apt-get update  
   
$ sudo apt-get install \  
 ca-certificates \  
 curl \  
 gnupg \  
 lsb-release  
$ sudo mkdir -p /etc/apt/keyrings  
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg  
$ echo \  
 "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \  
 $(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null  
   
$ sudo apt-get update  
$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin  
$ sudo groupadd docker  
$ sudo usermod -aG docker $USER

### Step 3: Verify Docker Installation

* Check if docker is successfully installed in your system by trying to run a container with the following command:

$ sudo docker run hello-world

To more detailing on the installation of docker on ubuntu, refer this – [Article](https://www.geeksforgeeks.org/how-to-install-and-configure-docker-in-ubuntu/)

Refere some videos and understand clear

Some example of Containerizing Application Using Docker

Sample Example to Push an image to Docker Hub

## **Sample Example: Containerizing Application Using Docker**

The following steps guides in containerizing the application using Docker:

### Step 1: Create Dokcerfile and Python Application

* Create a folder with 2 files (Dockerfile and main.py file) in it

### Step 2: Develop Python Code

* Edit main.py with the below code, or else you can try on developing your own python code.

Python

*#!/usr/bin/env python3*

print("Docker and GFG rock!")

### Step 3: Develop a Dockerfile

* Edit Dockerfile with the below instructions for developing the Dockerfile:

FROM python:latest  
COPY main.py /  
CMD [ "python", "./main.py" ]

### Step 4: Create a Docker Image

* Once you have created and edited the main.py file and the Dockerfile, create your image contain your application by running the following command:

$ sudo docker build -t python-test .

* The ‘-t’ option allows to define the name of your image. ‘python-test’ is the name we have chosen for the image.

### Step 5: Run Docker Container

* Once the image is created, your code is ready to launch.

$ sudo docker run python-test

To know more about this containerization of your application, refer this –[Article](https://www.geeksforgeeks.org/containerization-using-docker/)

## **Sample Example to Push an image to Docker Hub**

The following steps guides in pushing an image to the Dockerhub:

**Step 1:**Create an account on Docker Hub or use an existing one if you already have one.

**Step 2:**Click on the “Create Repository” button, put the name of the file, and click on “Create”.

**Step 3:**Now will “tag our image” and “push it to the Docker Hub repository” which we just created.

* Now, run the below command to list docker images:

$ docker images

* The above will give us this result

REPOSITORY TAG IMAGE\_ID CREATED SIZE afrozchakure/python-test latest c7857f97ebbd 2 hours ago 933MB

* Image ID is used to tag the image. The syntax to tag the image is:

docker tag <image-id> <your dockerhub username>/python-test:latest  
$ docker tag c7857f97ebbd afrozchakure/python-test:latest

**Step 4:**Push image to Docker Hub repository

$ docker push afrozchakure/python-test

To know more about publishing the images to dockerhub, refer this – [Article](https://www.geeksforgeeks.org/docker-publishing-images-to-docker-hub/)

## **Fetch and run the image from Docker Hub**

**1.**To remove all versions of a particular image from our local system, we use the Image ID for it.

$ docker rmi -f af939ee31fdc

**2.**Now run the image, it will fetch the image from the docker hub if it doesn’t exist on your local machine.

$ docker run afrozchakure/python-test

## **Alternatives of Docker**

The following are the alternatives of Docker:

* **Podman**: Offers a Docker-compatible container engine with a focus on security and compatibility, ideal for environments where Docker is not preferred or available.
* **rkt:** A lightweight container runtime developed by CoreOS, designed for simplicity, security, and composability, offering an alternative to Docker’s container runtime.
* **LXC (Linux Containers):** Provides operating-system-level virtualization for running multiple isolated Linux systems (containers) on a single host, offering a lightweight alternative to Docker for certain use cases.
* **containerd:** An industry-standard core container runtime developed by Docker, Inc., offering a minimal and stable platform for building containerized applications, often used as a lower-level alternative to Docker for more advanced container orchestration systems like Kubernetes.

## **Docker Security**

The following are the some of the insights on docker security:

* **Isolation**: Docker containers provides the strong isolation ensuring the applications and processing its running.
* **Immutable Infrastructure**: It promotes the use of immutable infrastructure, where containers are build from the immutable images that are version controlled.
* **Resource Constraints:**It allows you to define the resource constraints for containers such as CPU and memory limits.
* **Security Scanning:**It provides the built-in security scanning tools that allows you to scan the container images for known vulnerabilities and malware before deployment.

### What is Docker Hub in short?

*Docker Hub is a public registry where user/developers can store and share system images created by them and making them to easily access and reuse images for other software developers.*

### What are Docker Logs?

*Docker daemon and Docker containers will generate the docker logs in the form of text messages which will helps you in further to troubleshoot problems, monitor the performance of your applications, and gather information about the state of your Docker environment.*

### What is Docker Build?

*Docker build is an command which is used to build the docker image by using the Dockerfile.*

### What is Docker Ubuntu?

*“Docker Ubuntu” is the term used for utilizing the Docker service with Ubuntu-based operating systems. Running Docker is best suited for the popular Linux distribution Ubuntu.*

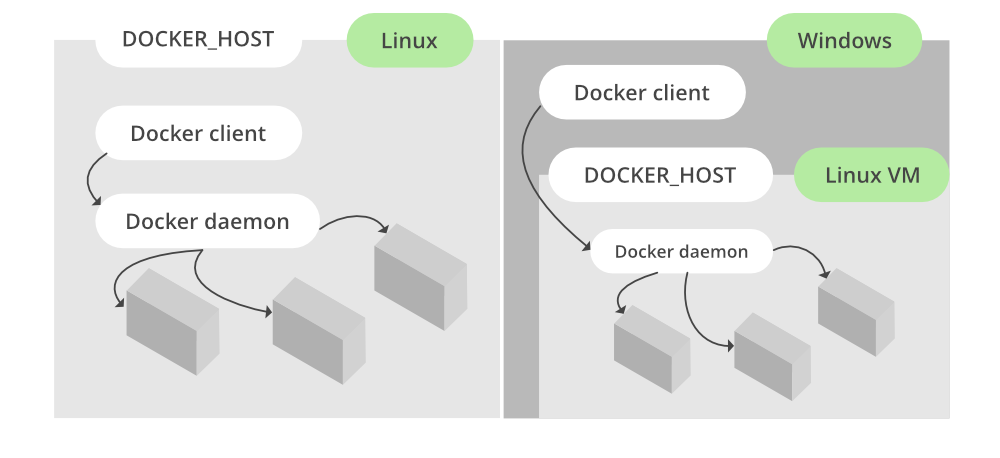
# Docker – Installation on Windows

Last Updated : 14 Apr, 2022

In this article, we are going to see how to install Docker on Windows.

*On windows if you are not using operating system Windows 10 Pro then you will have to install our docker toolbox and here docker will be running inside a virtual machine and then we will interact with docker with a docker client*

To understand this let us look at the left side of the below image where you can see on Linux operating system docker is installed directly on the operating system but if you look at the right side on Windows operating system docker is installed inside a Linux virtual machine and we use a docker client to interact with docker.



### Prerequisites:

* Windows 7 or higher
* 64-bit Operating System
* Download and Install[Linux kernel update package](https://wslstorestorage.blob.core.windows.net/wslblob/wsl_update_x64.msi)

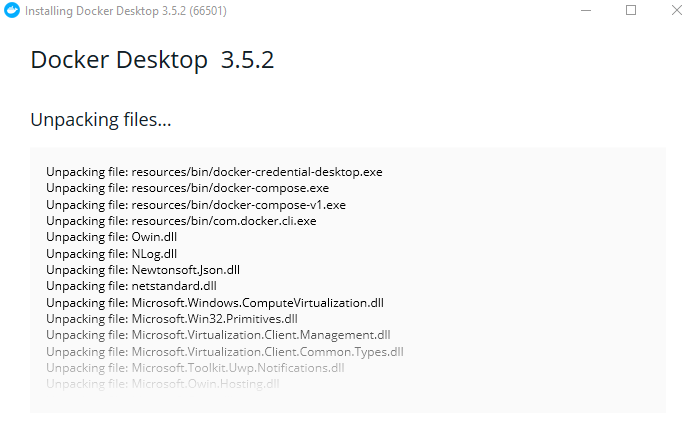
### Installation:

Follow the below steps to install Docker on Windows operating system:

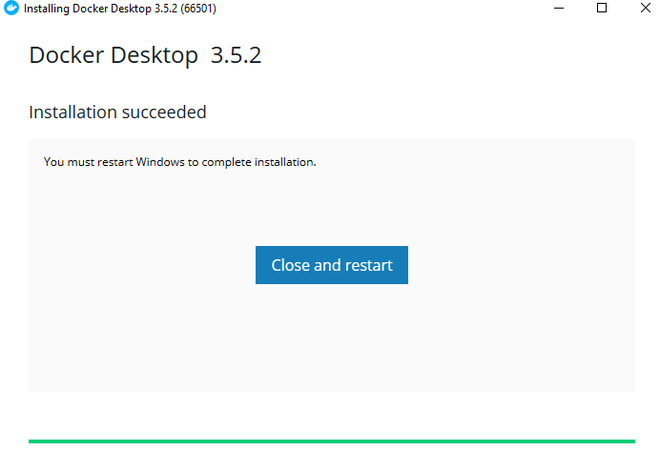
**Step 1:** Visit this website and click on the **“Docker Desktop for Windows”** button.



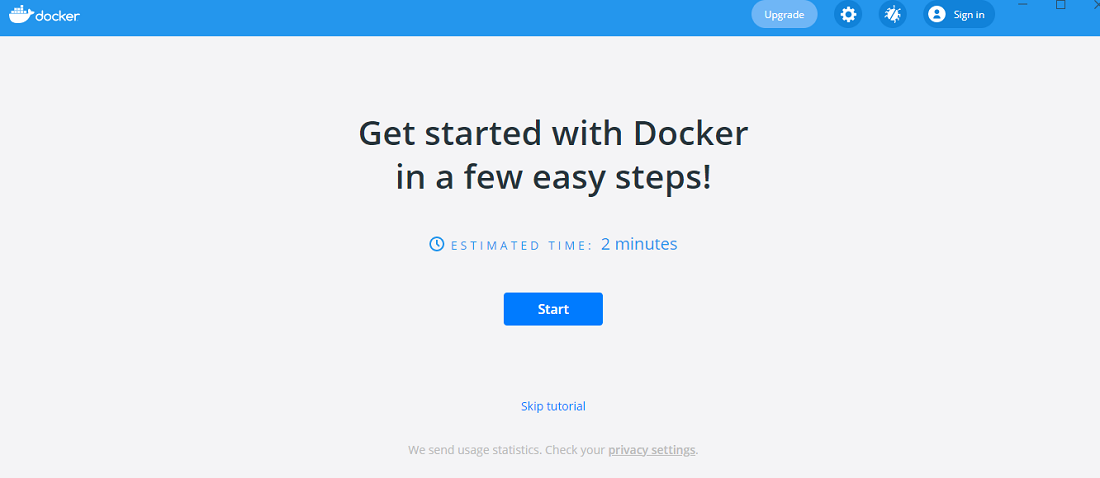
**Step 2:**The installer will ask you to install WSL2, so click on the tick box next to it and click on ok. This will download and install WSL2 for you as shown below:



**Step 3:**Then we will need to restart our system, so click on “**Close and restart”** button in the Docker installer window.



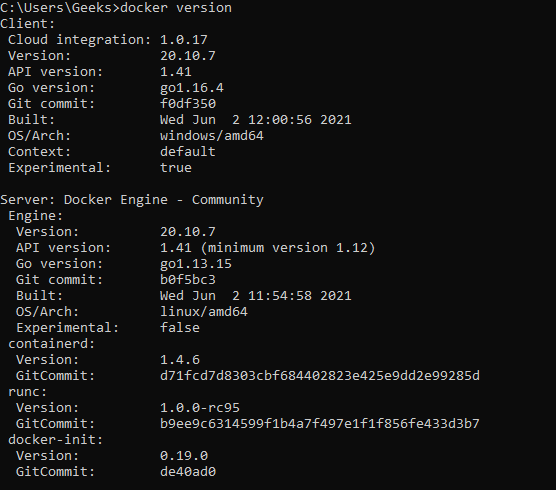
After you restart the computer it will open up the Docker windows as shown below:



At this point we have successfully installed docker on our Windows operating system. We can verify this using the below command:

docker version

This will result in the following informations:



# Docker Tutorial

Last Updated : 22 Jun, 2024

n this Docker Tutorial, you’ll learn all the basic to advanced concepts like Docker installation, Docker container, Docker commands, Docker run, Docker images, Docker compose, Docker engine, Docker networking, etc.

Docker is a powerful tool used for developing, packaging, and deploying applications efficiently. Docker is a container management service. Docker was released in 2013. It is open-source and available for different platforms like Windows, macOS, and Linux. Docker is quickly shipping, testing, and deploying code.

So that it reduces your delay between writing code and running it in production. You can create self-contained environments known as containers. That can run consistently on different platforms.



You need to install the Docker engine on your computer or device. The Docker engine allows you to create and manage docker containers, docker images, docker hub, docker desktop, etc.

**Table of Content**

* [Introduction](https://www.geeksforgeeks.org/docker-tutorial/?ref=lbp#int)
* [Docker Installation](https://www.geeksforgeeks.org/docker-tutorial/?ref=lbp#inst)
* [Docker Commands](https://www.geeksforgeeks.org/docker-tutorial/?ref=lbp#comm)
* [Docker Run](https://www.geeksforgeeks.org/docker-tutorial/?ref=lbp#run)
* [Docker Images](https://www.geeksforgeeks.org/docker-tutorial/?ref=lbp#img)
* [Docker Compose](https://www.geeksforgeeks.org/docker-tutorial/?ref=lbp#com)
* [Docker Engine, Storage](https://www.geeksforgeeks.org/docker-tutorial/?ref=lbp#eng)
* [Docker Networking](https://www.geeksforgeeks.org/docker-tutorial/?ref=lbp#net)
* [Docker Registry](https://www.geeksforgeeks.org/docker-tutorial/?ref=lbp#reg)
* [Docker Containers and Managing Containers](https://www.geeksforgeeks.org/docker-tutorial/?ref=lbp#con)
* [Docker Swarm](https://www.geeksforgeeks.org/docker-tutorial/?ref=lbp#swa)
* [Miscellaneous](https://www.geeksforgeeks.org/docker-tutorial/?ref=lbp#miscellaneous)
* [Why to learn Docker?](https://www.geeksforgeeks.org/docker-tutorial/?ref=lbp#why-to-learn-docker)
* [Features of Docker](https://www.geeksforgeeks.org/docker-tutorial/?ref=lbp#features-of-docker)
* [Advantages of Docker](https://www.geeksforgeeks.org/docker-tutorial/?ref=lbp#advantages-of-docker)
* [Disadvantages of Docker](https://www.geeksforgeeks.org/docker-tutorial/?ref=lbp#disadvantages-of-docker)

## Introduction

If you are here, then it very obvious that you want to learn the Docker, so here in this section we have listed all the basic intro of Docker.

* [Introduction to Docker](https://www.geeksforgeeks.org/introduction-to-docker/)
* [Features of Docker](https://www.geeksforgeeks.org/features-of-docker/)
* [Docker Architecture](https://www.geeksforgeeks.org/architecture-of-docker/)
* [Docker Hub](https://www.geeksforgeeks.org/what-is-docker-hub/?ref=rp)
* [Docker Cloud](https://www.geeksforgeeks.org/what-is-docker-cloud/)

## Docker Installation

This section will walk you through the steps of installing Docker on your system, whether it’s Windows, macOS, or Linux. We’ll cover the different installation methods, system requirements, and post-installation configurations to ensure you have Docker up and running smoothly, ready to build and run your containerized applications.

* [Docker Installation on Windows](https://www.geeksforgeeks.org/docker-installation-on-windows/)
* [Docker Installation on Windows using Chocolatey](https://www.geeksforgeeks.org/how-to-install-docker-using-chocolatey-on-windows/)
* [Docker Installation on Ubuntu](https://www.geeksforgeeks.org/how-to-install-and-configure-docker-in-ubuntu/)
* [Docker Installation on macOS](https://www.geeksforgeeks.org/how-to-install-docker-on-macos/)
* [Docker Installation on Arch-based Linux](https://www.geeksforgeeks.org/how-to-install-and-configure-docker-on-arch-based-linux-distributionsmanjaro/)
* [Docker Installation on RedHat](https://www.geeksforgeeks.org/how-to-install-docker-ce-in-redhat-8/)

## Docker Commands

In this part of the Docker tutorial we will dives into the essential commands you need to navigate the world of Docker. Whether you’re building, running, or managing containers, this guide equips you with the tools to control your containerized applications.

* [Docker Instructions](https://www.geeksforgeeks.org/docker-instruction-commands/)
* [Run Commands Inside Containers](https://www.geeksforgeeks.org/running-commands-inside-docker-container/)
* [USER Instruction](https://www.geeksforgeeks.org/docker-user-instruction/)

## Docker Run

Docker Run refers to the command used in Docker to create and start containers based on Docker images. It’s a fundamental aspect of working with Docker, allowing users to deploy applications and services quickly and efficiently within isolated environments known as containers. Here in this section, you will get to know all the details about the Docker Run.

* [Dockerfile](https://www.geeksforgeeks.org/docker-concept-of-dockerfile/)
* [Syntax of Dockerfile](https://www.geeksforgeeks.org/what-is-dockerfile-syntax/)
* [How to Build a Web Server Docker File?](https://www.geeksforgeeks.org/how-to-build-a-web-server-docker-file/)

## **Docker Images**

A Docker Image contains everything a container needs to run, including the application code, libraries, dependencies, and the operating system it needs.

* [Docker Images](https://www.geeksforgeeks.org/what-is-docker-images/)
* [Working with Images](https://www.geeksforgeeks.org/working-with-docker-images/)
* [Publish Docker Images in Hub](https://www.geeksforgeeks.org/docker-publishing-images-to-docker-hub/?ref=rp)
* [Create a Customized Image](https://www.geeksforgeeks.org/how-to-create-your-own-docker-customised-image/)
* [Use Image Tags](https://www.geeksforgeeks.org/docker-using-image-tags/)
* [How to use Next.js Image](https://www.geeksforgeeks.org/next-js-docker-images/)
* [How to use Local Image with Minikube](https://www.geeksforgeeks.org/how-to-use-local-docker-images-with-minikube/)

## Docker Compose

Managing multiple containers can get complex! Docker Compose simplifies this process. In this section we have listed down all the details about the Docker Compose like intro and Compose tools.

* [Introduction to Docker Compose](https://www.geeksforgeeks.org/docker-compose/)
* [Compose tools to run Multi Container Apps](https://www.geeksforgeeks.org/docker-compose-tool-to-run-multi-container-applications/)

## Docker Engine, Storage

Docker Engine, also known as Docker Daemon, is the core component of the Docker platform responsible for running and managing Docker containers. Explore this section to get all details about Docker Engine.

* [Docker Storage](https://www.geeksforgeeks.org/data-storage-in-docker/)
* [Docker Data Storage](https://www.geeksforgeeks.org/data-storage-in-docker/)
* [Backup Docker Container](https://www.geeksforgeeks.org/backing-up-a-docker-container/)
* [Manage Volumes using CLI](https://www.geeksforgeeks.org/using-cli-to-manage-docker-volumes/)

## Docker Networking

Docker Networking refers to the set of mechanisms and technologies Docker provides for communication between Docker containers, as well as between containers and the outside world. Go through this section to get more details about Docker Networking.

* [Docker Networking](https://www.geeksforgeeks.org/basics-of-docker-networking/)
* [Docker Ports](https://www.geeksforgeeks.org/docker-managing-ports/)
* [Creating a Network and connecting a Container](https://www.geeksforgeeks.org/creating-a-network-in-docker-and-connecting-a-container-to-that-network/)
* [Connecting Two Docker Containers Over the Same Network](https://www.geeksforgeeks.org/connecting-two-docker-containers-over-the-same-network/)
* [Default Bridge Networking](https://www.geeksforgeeks.org/how-to-use-docker-default-bridge-networking/)
* [Create your own secure Home Network using Pi-hole and Docker](https://www.geeksforgeeks.org/create-your-own-secure-home-network-using-pi-hole-and-docker/)

## Docker Registry

After knowing the Docker Networking, in Docker tutorial, in this section we are going to discuss Docker Registry which is refer as central repository for storing the and managing Docker image.

* [Docker Registry](https://www.geeksforgeeks.org/what-is-docker-registry/)
* [Docker – Public Repositories](https://www.geeksforgeeks.org/docker-using-public-repositories-to-host-docker-images/)
* [Docker – Private Registries](https://www.geeksforgeeks.org/docker-private-registries/)
* [Creating a Private Repository and Push an Image to That Private Repository](https://www.geeksforgeeks.org/creating-a-private-repository-and-push-an-image-to-that-private-repository/)
* [Using Public Repositories To Host Docker Images](https://www.geeksforgeeks.org/docker-using-public-repositories-to-host-docker-images/)

## Docker Containers and Managing Containers

Explores this section to get to know about the fundamental concepts and practical aspects of utilizing Docker technology to deploy and manage software applications efficiently in this section.

* [Containerization using Docker](https://www.geeksforgeeks.org/containerization-using-docker/)
* [Docker Container Virtualization](https://www.geeksforgeeks.org/virtualisation-with-docker-containers/)
* [Docker Container for Node.js](https://www.geeksforgeeks.org/docker-docker-container-for-node-js/?ref=rp)
* [Remove Containers](https://www.geeksforgeeks.org/remove-all-containers-and-images-in-docker/)
* [Push a Container Image to a Docker Repository](https://www.geeksforgeeks.org/how-to-push-a-container-image-to-a-docker-repository/)
* [Docker Container Linking](https://www.geeksforgeeks.org/docker-container-linking/)
* [Manage Containers using CLI](https://www.geeksforgeeks.org/tips-to-manage-docker-containers-using-cli/)
* [Mount Volume inside Container](https://www.geeksforgeeks.org/mounting-a-volume-inside-docker-container/)
* [Difference between Images and Container](https://www.geeksforgeeks.org/difference-between-docker-image-and-container/)
* [Difference between Virtual Machines and Containers](https://www.geeksforgeeks.org/difference-between-virtual-machines-and-containers/)
* [How to Install Linux Package in Docker Container](https://www.geeksforgeeks.org/how-to-install-linux-packages-inside-a-docker-container/)
* [Copying Files to and from Docker Containers](https://www.geeksforgeeks.org/copying-files-to-and-from-docker-containers/)
* [Run MongoDB as Container](https://www.geeksforgeeks.org/how-to-run-mongodb-as-a-docker-container/)
* [Container for Node.js](https://www.geeksforgeeks.org/docker-docker-container-for-node-js/)
* [Container for NGNIX](https://www.geeksforgeeks.org/docker-container-for-nginx/)
* [How to Provide the Static IP to a Docker Container?](https://www.geeksforgeeks.org/how-to-provide-the-static-ip-to-a-docker-container/)

## Docker Swarm

Docker Swarm steps in as your reliable organizer, making sure all your containers are in the right place and working well together. Go through this section to get an all about Docker Swarm.

* [Introduction to Docker Swarm](https://www.geeksforgeeks.org/introduction-to-docker-swarm-mode/?ref=rp)
* [Difference between Kubernetes and Docker Swarm](https://www.geeksforgeeks.org/difference-between-kubernetes-and-docker-swarm/)

## Miscellaneous

* [Running a Java Application using Docker](https://www.geeksforgeeks.org/containerizing-java-applications-creating-a-spring-boot-app-using-dockerfile/)
* [Running a PHP Application using Docker](https://www.geeksforgeeks.org/how-to-create-a-php-docker-container/)
* [Running a Python Application using Docker](https://www.geeksforgeeks.org/how-to-run-a-python-script-using-docker/)
* [How to Dockerize an ExpressJS App?](https://www.geeksforgeeks.org/how-to-dockerize-an-expressjs-app/)
* [How to Dockerize a ReactJS App?](https://www.geeksforgeeks.org/how-to-dockerize-a-reactjs-app/)
* [Docker Vs Kubernetes](https://www.geeksforgeeks.org/kubernetes-vs-docker/)
* [Deploying WebApps on Docker](https://www.geeksforgeeks.org/docker-deploying-webapps-on-docker/)
* [Docker Continuous Integration](https://www.geeksforgeeks.org/docker-continuous-integration/)
* [Difference Between Vagrant and Docker](https://www.geeksforgeeks.org/difference-between-vagrant-and-docker/)
* [How to Setup Jenkins in Docker Container?](https://www.geeksforgeeks.org/how-to-setup-jenkins-in-docker-container/)

**Docker Container**: A Docker container is a lightweight and executable package of software. It includes everything used to run an application, code, libraries, and dependencies. You don’t need to allocate any memory for the application. It can automatically generate space according to the requirements.

## Why to learn Docker?

A couple of years back, when organizations needed other applications, they buy a server without knowing the performance requirement of the software/application. This results waste of money and resources. Then Virtual machines come onto the market, which allows engineers to run multiple applications on the same resource but as a completely different server. But Every application to run on a Virtual machine requires OS and every OS needs its own CPU, RAM, etc. to run. Which increases the cost.

Then Docker Model comes into the scope, which overcomes the drawbacks of the Virtual Machine. It reduces the wastage of resources by sharing OS, memory, and CPU, and It offers many benefits for developers and system administrators, like consistency, portability, efficiency, security, scalability, and version control. These benefits make it easier to manage and deploy applications.

## **Advantages of Docker**

1. **Light weight**
2. **Developer productivity**
3. **Increased application portability**
4. **Efficient resource utilization**
5. **Simplified deployment**
6. **Scaling of application**

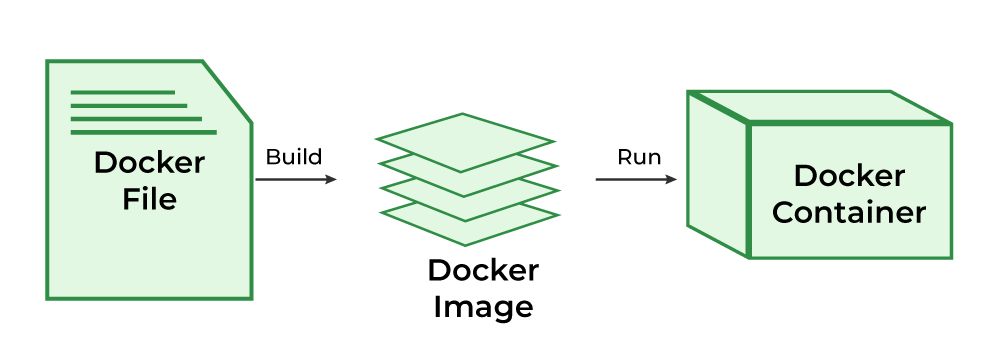
## **Disadvantages of Docker**

* Complexity will increase due to layering.
* It is difficult to manage a large number of containers.
* For an application that needs better graphics, Docker is not suitable for it.
* Cross-platform compatibility is not allowed.

# What is Dockerfile?

Last Updated : 24 Jan, 2024

The operating system (OS) libraries and dependencies required to run the application source code which is not reliant on the underlying operating system (OS) included in the Dockerfile, which is a standardized, executable component. Programmers may design, distribute, launch, run, upgrade, and manage containers using the open-source platform Docker. Enterprise Edition (EE) and Community Edition (CE) of Docker are both available. The Enterprise Version is for businesses and IT teams working on mission-critical production applications, while the Community Edition is suitable for small teams just learning Docker.



**Table of Content**

* [What is Docker?](https://www.geeksforgeeks.org/what-is-dockerfile/#what-is-docker)
* [What is Dockerfile?](https://www.geeksforgeeks.org/what-is-dockerfile/#what-is-dockerfile)
* [What is Docker Image?](https://www.geeksforgeeks.org/what-is-dockerfile/#what-is-docker-image)
* [What is Docker Container?](https://www.geeksforgeeks.org/what-is-dockerfile/#what-is-docker-container)
* [Dockerfile commands/Instructions](https://www.geeksforgeeks.org/what-is-dockerfile/#dockerfile-commandsinstructions)
* [Steps To Create a Dockerfile](https://www.geeksforgeeks.org/what-is-dockerfile/#steps-to-create-a-dockerfile)
* [Example 1: Steps To Create Dockerfile With Example(Jenkins)](https://www.geeksforgeeks.org/what-is-dockerfile/#example-1-steps-to-create-dockerfile-with-examplejenkins)
* [Dockerfile for Jenkins image](https://www.geeksforgeeks.org/what-is-dockerfile/#dockerfile-for-jenkins-image)
* [Example 2: Steps To Create Dockerfile](https://www.geeksforgeeks.org/what-is-dockerfile/#example-2-steps-to-create-dockerfile)
* [Conclusion](https://www.geeksforgeeks.org/what-is-dockerfile/#conclusion)
* [Dockerfile – FAQs](https://www.geeksforgeeks.org/what-is-dockerfile/#dockerfile-faqs)

## What is Docker?

[Docker](https://www.geeksforgeeks.org/introduction-to-docker/) is an open-source platform where developers can containerize the application. Containers are accessible before Docker but have gained popularity as a result of Docker. The most crucial aspects of Docker are the Docker Engine and [Docker Hub](https://www.geeksforgeeks.org/what-is-docker-hub/). The first one works on your local system to run your program, and the second one is similar to a cloud service where we can share our docker images with everyone.

## **What is Docker Image?**

An artifact with several layers and a lightweight, compact stand-alone executable package that contains all of the components required to run a piece of software, including the code, a runtime, libraries, environment variables, and configuration files is called a [Docker image](https://www.geeksforgeeks.org/what-is-docker-images/).

## **Dockerfile commands/Instructions**

### ****1. FROM****

Represents the base image(OS), which is the command that is executed first before any other commands.

#### **Syntax:**

*FROM <ImageName>*

**Example:**The base image will be ubuntu:19.04 [Operating System](https://www.geeksforgeeks.org/operating-systems/).

FROM ubuntu:19.04

### ****2. COPY****

The copy command is used to copy the file/folders to the image while building the image.

#### **Syntax:**

*COPY <Source> <Destination>*

**Example:** Copying the .war file to the [Tomcat](https://www.geeksforgeeks.org/how-to-install-apache-tomcat-8-on-linux/) webapps directory

COPY target/java-web-app.war /usr/local/tomcat/webapps/java-web-app.war

### ****3. ADD****

While creating the image, we can download files from distant HTTP/HTTPS destinations using the ADD command.

#### **Syntax:**

*ADD <URL>*

**Example:**Try to download Jenkins using ADD command

ADD https://get.jenkins.io/war/2.397/jenkins.war

### ****4. RUN****

Scripts and commands are run with the RUN instruction. The execution of RUN commands or instructions will take place while you create an image on top of the prior layers (Image).

#### **Syntax:**

*RUN < Command + ARGS>*

**Example:**

RUN touch file

### ****5. CMD****

The main purpose of the CMD command is to start the process inside the container and it can be overridden.

#### **Syntax:**

*CMD [command + args]*

**Example:**Starting [Jenkins](https://www.geeksforgeeks.org/what-is-jenkins/)

CMD ["java","-jar", "Jenkins.war"]

### ****6. ENTRYPOINT****

A container that will function as an executable is configured by ENTRYPOINT. When you start the Docker container, a command or script called ENTRYPOINT is executed. It can’t be overridden.The only difference between [CMD and ENTRYPOINT](https://www.geeksforgeeks.org/difference-between-run-vs-cmd-vs-entrypoint-docker-commands/) is CMD can be overridden and ENTRYPOINT can’t.

#### **Syntax:**

*ENTRYPOINT [command + args]*

**Example:** Executing the **echo command.**

ENTRYPOINT["echo","Welcome to GFG"]

### ****7. MAINTAINER****

By using the MAINTAINER command we can identify the author/owner of the Dockerfile and we can set our own author/owner for the image.

#### **Syntax:**

*MAINTAINER <NAME>*

**Example:** Setting the author for the image as a GFG author.

MAINTAINER GFG author

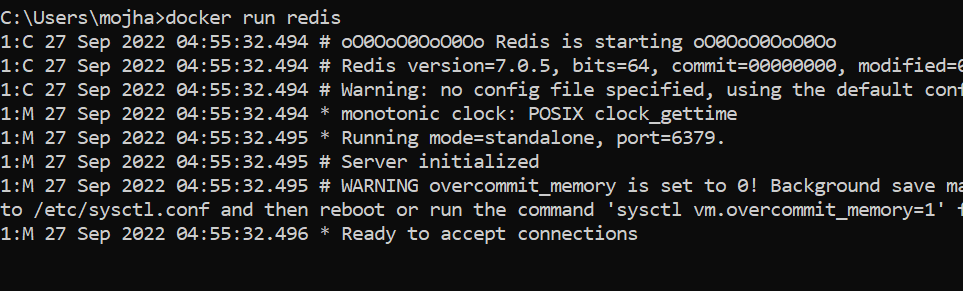
To know more the sytnx of Dokcerfile refer to the [Syntax of Dockerfile](https://www.geeksforgeeks.org/what-is-dockerfile-syntax/).

## **Docker Commands**

### ****Docker Run command****

This command is used to run a container from an image. The docker run command is a combination of the docker create and docker start commands. It creates a new container from the image specified and starts that container. if the [docker image](https://www.geeksforgeeks.org/what-is-docker-images/) is not present, then the docker run pulls that.

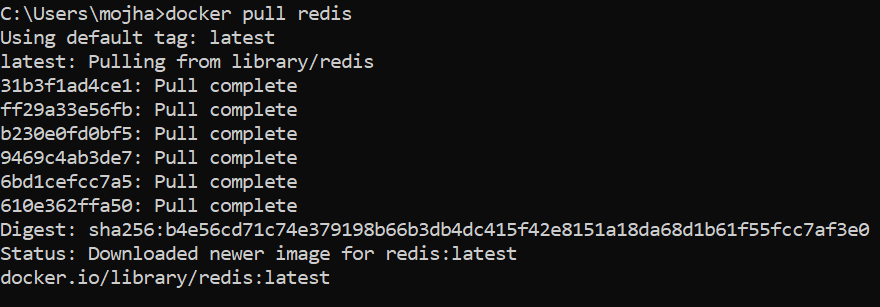
$ docker run <image\_name>  
To give name of container  
$ docker run --name <container\_name> <image\_name>



### ****Docker Pull****

This command allows you to pull any image which is present in the official [registry of docker](https://www.geeksforgeeks.org/what-is-docker-registry/), [Docker hub](https://www.geeksforgeeks.org/what-is-docker-hub/). By default, it pulls the latest image, but you can also mention the version of the image.

$ docker pull <image\_name>

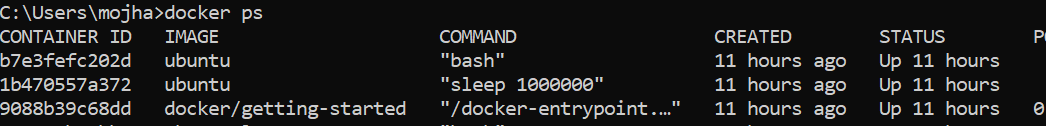


### ****Docker PS****

This command (by default) shows us a list of all the running containers. We can use various flags with it.

* **-a flag:** shows us all the containers, stopped or running.
* **-l flag:**shows us the latest container.
* **-q flag**: shows only the Id of the containers.

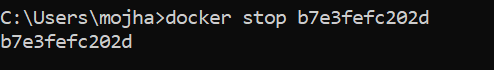
$ docker ps [options..]



### ****Docker Stop****

This command allows you to stop a container if it has crashed or you want to switch to another one.

$ docker stop <container\_ID>



### ****Docker Start****

Suppose you want to start the stopped container again, you can do it with the help of this command.

$ docker start <container\_ID>

### ****Docker rm****

To delete a container. By default when a container is created, it gets an ID as well as an imaginary name such as confident\_boyd, heuristic\_villani, etc. You can either mention the container name or its ID.

Some important flags:

* **-f flag:**remove the container forcefully.
* **-v flag:**remove the volumes.
* **-l flag:**remove the specific link mentioned.

$ docker rm {options} <container\_name or ID>

docker remove an image

### Docker RMI

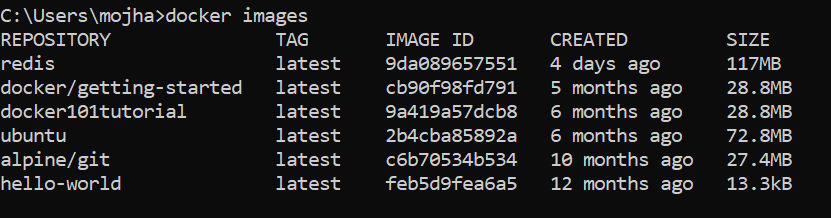
To delete the image in docker. You can delete the images which are useless from the docker local storage so you can free up the space

docker rmi <image ID/ image name>

### ****Docker Images****

Lists all the pulled images which are present in our system.

$ docker images



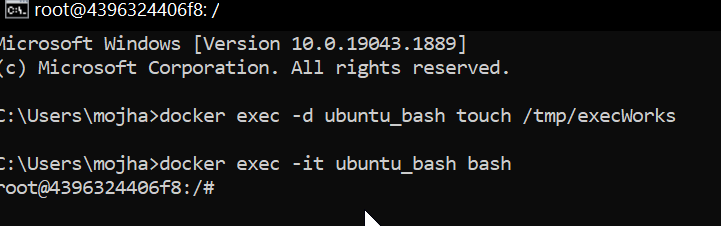
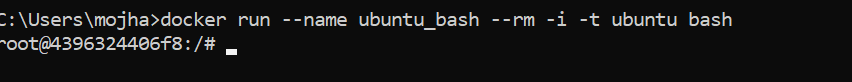
### ****Docker exec****

This command allows us to run new commands in a running container. This command only works until the container is running, after the container restarts, this command does not restart.

Some important flags:

* **-d flag:**for running the commands in the background.
* **-i flag:**it will keep STDIN open even when not attached.
* **-e flag:**sets the environment variables

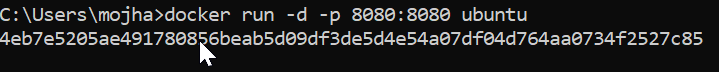
$ docker exec {options}



### ****Docker Ports (Port Mapping)****

In order to access the [docker container](https://www.geeksforgeeks.org/containerization-using-docker/) from the outside world, we have to map the port on our host( Our laptop for example), to the port on the container. This is where port mapping comes into play.

$ docker run -d -p <port\_on\_host>   
<port\_on\_container> Container\_name



So these were the 9 most basic docker commands that every beginner must know. Containerization is a very vast topic but you can start from the very basic commands and by practicing them daily you can master them.

### Docker Login

The Docker login command will help you to authenticate with the Docker hub by which you can push and pull your images.

docker login

It will ask you to enter the username and password after that you will authenticate with DockerHub and you can perform the tasks.

### Docker Push

Once you build your own customized image by using Dockerfile you need to store the image in the remote registry which is DockerHub for that you need to push your image by using the following command.[To know more about How to Push a Container Image to a Docker Repository?](https://www.geeksforgeeks.org/how-to-push-a-container-image-to-a-docker-repository/)

docker push <Image name/Image ID>

### Docker Build

The docker build command is used to build the docker images with the help of [Dockerfile.](https://www.geeksforgeeks.org/docker-concept-of-dockerfile/)

docker build -t image\_name:tag .

In the place of **image\_name** use the name of the image you build with and give the **tag number** and **. “dot”**represents the current directory.

### Docker Stop

You can stop and start the docker containers where you can do the maintenance for containers. To stop and start specific containers you can use the following commands.

docker stop container\_name\_or\_id

### Stop Multiple Containers

Instead of stopping a single container. You can stop multiple containers at a time by using the following commands.

docker stop container1 container2 container3

### Docker Restart

While running the containers in Docker you may face some errors and containers fails to start. You can restart the containers to resolve the containers by using the following commands.

docker restart container\_name\_or\_id

### Docker Inspection

Docker containers will run into some errors in real time to debug the container’s errors you can use the following commands.

docker inspect container\_name\_or\_id

### Docker Commit command

After running the containers by using the current image you can make the updates to the containers by interacting with the containers from that containers you can create an image by using the following commands.

docker commit container\_name\_or\_id new\_image\_name:tag

## **Docker Basic Command**

Following are the some of the docker basic commands

1. **docker images:** Docker images will list all the images which are pulled or build in that docker host.
2. **docker pull:**Docker pull will the docker images from the dockerhub.
3. **docker run:**Docker run will run the docker image as an container.
4. **docker ps:**Docker run will list all the containers which are running in the docker host.
5. **docker stop:**Docker stop will stop the docker container which are already running.
6. **docker rm:**Docker rm command will remove the containers which are in the stop condition.

## Docker Commands List

Following are the docker commands which listed form build and Docker image to running it an Docker container and attaching the docker volumes to it.

### Docker Image Command

1. **docker build command:** It will build Docker images by using the **Dockerfile.**
2. **docker pull command:** Docker pull command will pull the **Docker image** whcih is avalible in the **dockerhub.**
3. **docker images command:**It will list all the images which are pulled and build in the docker host.
4. **docker inspect command:**It will helps to debug the docker image if any errors occurred while building an image or pulling the image.
5. **docker push command:**Docker command will push the docker image into the Dockerhub.
6. **docker save command:**It will save the docker image in the form of dockerfile.
7. **docker rmi command:**It will remove the docker image.

### Docker Container Command

1. **docker attach command:** Connecting to an Existing Container.
2. **docker ps command:**To list the running containers.
3. **docker container inspect infinite Command:**To Inspect the Docker containers.
4. **docker exec command:**To execute the commands in the running containers.
5. **docker cp command:** To copy the file from docker host to the docker containers,

### 2. Docker Command Not Found

*Docker command not found error will get if the docker cli was not installed in your system.*

### 3. Docker Command To list the All Containers

*To list all the containers in the docker you can use the****docker pa -a.***

### 4. Docker Command To Build An Image

*To build the docker image you can use the following command.*

***docker build -t <image name>:<tag> <path to Dockerfile>***

### 5. Docker Command To Remove An Image

*Docker Command to remove the docker images was*

***docker rmi <image name>:<tag>***

### 6. Docker Command Cheat Sheet

*To read the docker cheat sheet refer to*[*Docker Cheat Sheet – Most Important Docker Commands.*](https://www.geeksforgeeks.org/docker-cheat-sheet/)

### 7. Docker Command to Command to check the Running Containers

*docker ps is the command to check the running containers.*

### 8. What is the Docker file with all commands?

*Dockerfile is set of instruction to build the Docker image*

## Create a Dockerfile

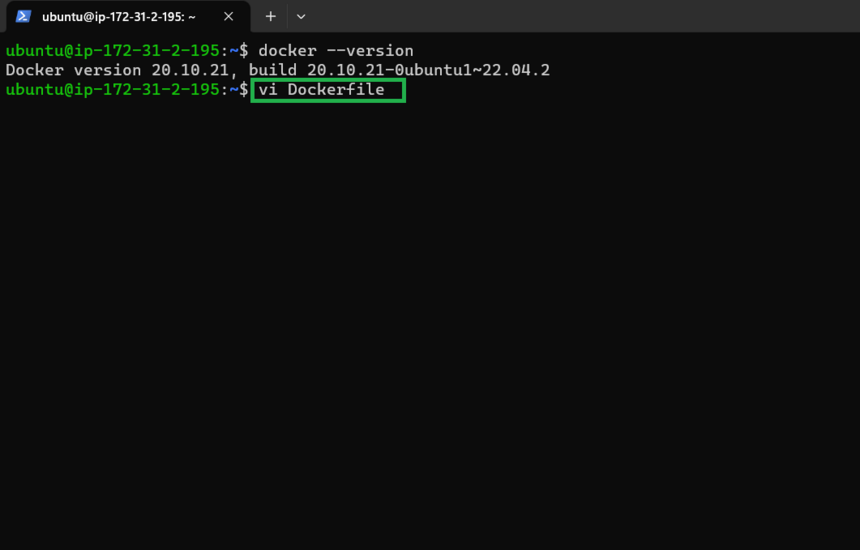
* Create a file named Dockerfile.
* Add instructions in Dockerfile.
* Build Dockerfile to create an image.
* Run the image to create a container.

## Example 1: Steps To Create Dockerfile With Example(Jenkins)

In this example, we will write the Dockerfile for Jenkins and build an image by using Dockerfile which has been written for Jenkins and we will run it as a container.

**Step 1:**Open Docker and create a file with the name **Dockerfile.**

**Step 2:**Open the Dockerfile by using the vi editor and start writing the command that is required to build the Jenkins image.



## **Dockerfile for Jenkins image**

We used [JDK](https://www.geeksforgeeks.org/jdk-in-java/) as a base image because Jenkins’s pre-requisite is JDK after that we added a command called**MAINTAINER** which indicates the author or owner of the docker file and we added the**ENV**variable where we set the path for the Jenkins and by using **RUN**command we are creating the path and by using **ADD** we are downloading the Jenkins and starting the **.war**file with the help of **CMD**command.

FROM openjdk:11-jdk

MAINTAINER GFG author

LABEL env=production

ENV apparea /data/app

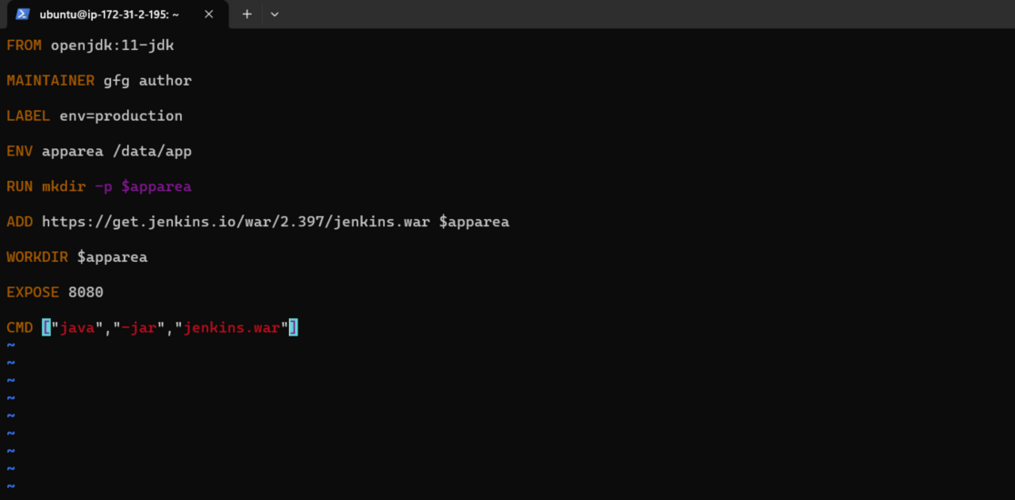
RUN mkdir -p $apparea

ADD <https://get.jenkins.io/war/2.397/jenkins.war> $apparea

WORKDIR $apparea

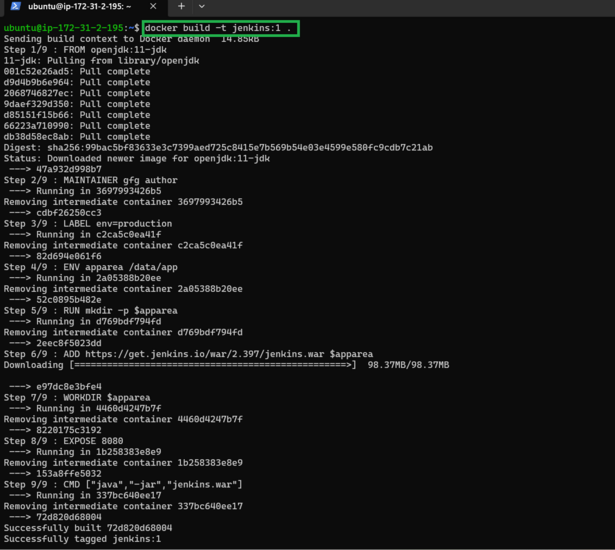
EXPOSE 8080

CMD ["java","-jar","jenkins.war"]



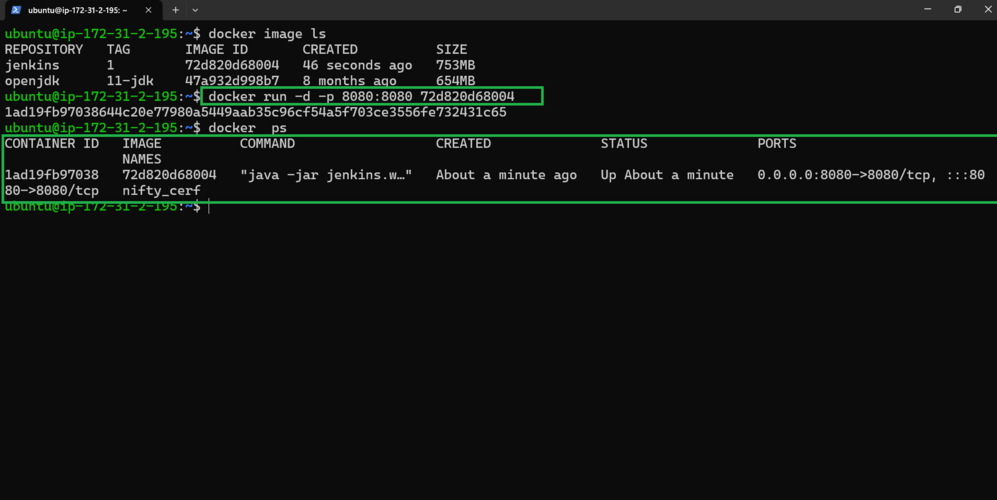
**Step 3:** Build the image by using the below command with the help of Dockerfile and give the necessary tags. and the dot(.) represents the current directory which is a path for Dockerfile.

docker build -t jenkins:1 .

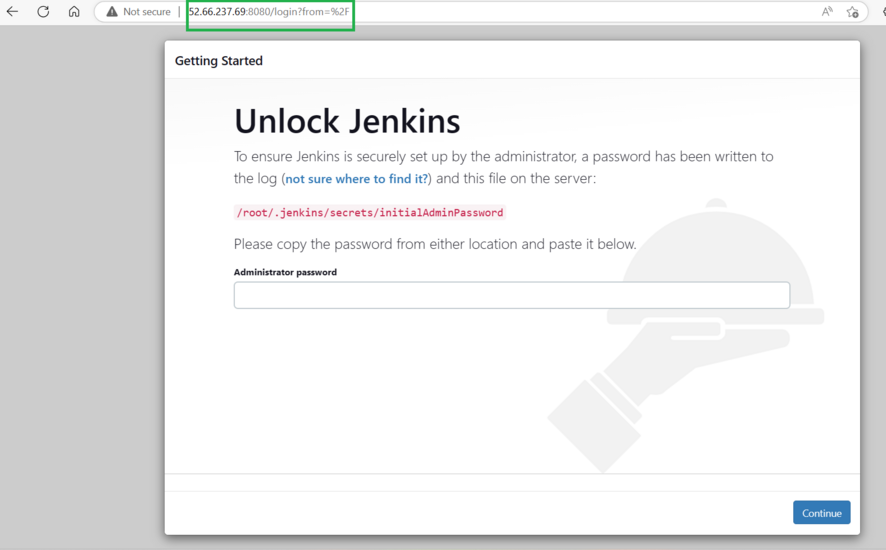


**Step 4:** Run the container with the help image ID or tag of the image by using the below command.

docker run -d -p 8080:8080 <Imagetag/ID>

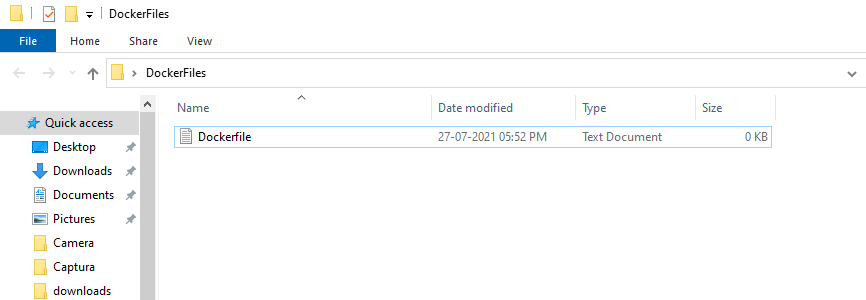


**Step 5:** Accesses the application (Jenkins) from the internet with the help of host port and hostIP (HostIP: Port)



## **Example 2: Steps To Create Dockerfile**

**Step 1:**Create a file name called **“Dockerfile”**.By default when you run the docker build commands docker searches for a file named Dockerfile. However, it is not compulsory, you can also give some different names, and then you can tell the docker that this particular file is local but for now we will go with the Dockerfile.



**Step 2:** The very first instruction that a docker file starts with is FROM. Here you have to give a base image. So for example, if you want to get a base image from Ubuntu we will use FROM Ubuntu.

FROM ubuntu

Then the other instruction is you have to give a **MAINTAINER.** This is optional but it’s a best practice that you give the maintainer of this image so that it is very easy to find out who is the maintainer and you can give your name and email as well. And if you want you can just give the email as well without giving the name. But here we are giving the entire thing.

MAINTAINER YOUR\_NAME <YOUR\_EMAIL\_ID>

Next, we want to run something so we will say run any command we can use **RUN**and add the command that you need to run.

RUN apt-get update

And if you want to run something on the command line during container creation you can give **CMD**and inside square brackets, and we add the command. Here it is as shown below:

CMD ["echo", "Hello Geeks!"]

At this point the file will have the following commands:

FROM ubuntu

MAINTAINER YOUR\_NAME <YOUR\_EMAIL\_ID>

RUN apt-get update

CMD ["echo", "Hello Geeks!"]

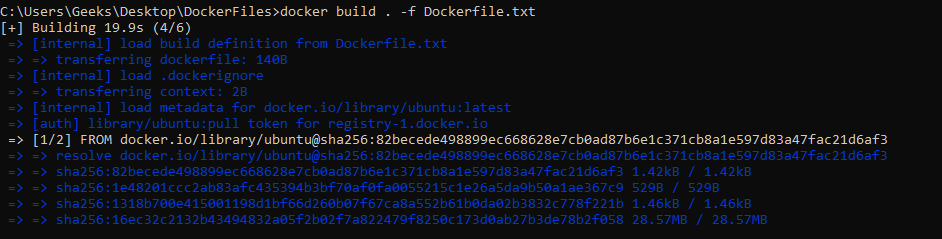
**Step 3:** Now we have to build the image so here are the commands you can use:

docker build /<FILE\_LOCATION>

Or,

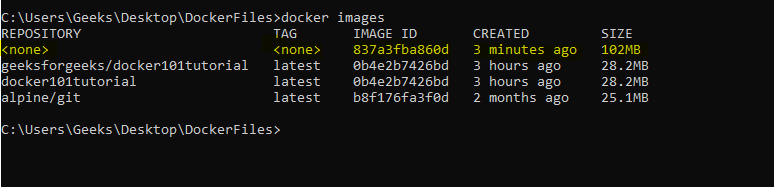
docker build . -f Dockerfile.txt

It says docker build and you have to give the location of your docker file.  This will start building the image.



Command to list the images

docker image ls / docker images



### What is Dockerfile Extension?

*Dockerfile is the source code of docker image and basically there is no extension for Dockerfile or you may use****“.dockerfile or .Dockerfile”****they can be used but they are not mandatory. The standard file name is Dockerfile.*

### Dockerfile Copy Directory

*The*[*COPY command*](https://www.geeksforgeeks.org/difference-between-the-copy-and-add-commands-in-a-dockerfile/)*or instruction in the dockerfile is used to copy the directories from docker host to the docker image.*

### Dockerfile vs Docker Compose

* ***Dockerfile:****Dockerfile consists of all the steps that are required to create docker image. Dockerfile consists of base image and the all remaining environmental variables.*
* [***Docker Compose***](https://www.geeksforgeeks.org/docker-compose/)***:****YAML file consists of multi-container applications. Simplifies the orchestration and configuration of Dockerized applications by specifying services, networks, and volumes.*

# Next.js Docker Images

Last Updated : 06 Feb, 2023

Next.js is a React framework that helps in building server-side rendering web applications.

**What Is Docker Image?**

In order to run these applications, we first need to create an **image** of the current application state. An image can be sometimes referred to as a snapshot of our project. Images are read-only in nature and consist of file that contains the source code, libraries, dependencies, tools, and other files needed for an application to run.

A Docker image is a read-only template that contains a set of instructions for creating a container that can run on the Docker platform.

To use a programming analogy, if an image is a class, then a container is an instance of a class. Containers are the reason why you’re using Docker. It is a lightweight and portable environment to run our applications.

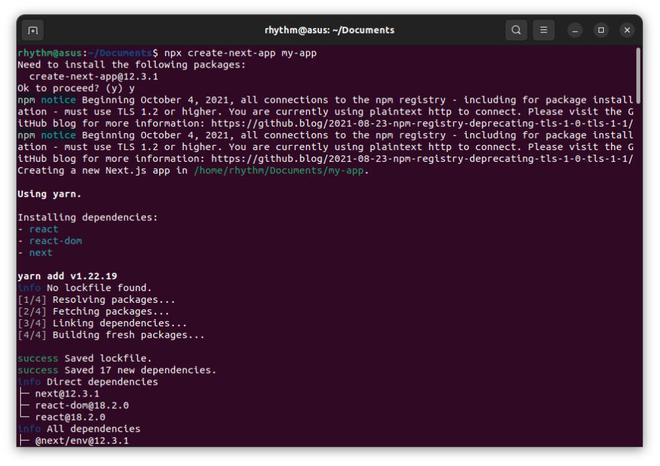
Containers are running instances of Docker images. Containers run the actual applications.

**Steps to initialize NextJs Docker Images:**Follow the below steps to initialize the NextJs Docker Images:

**Step 1:**Initializing NextJs project

Go to the directory where you want to initialize your project and use npx to download all required files.

$ npx create-next-app



**Step 2:**Open The Application On your code editor. I am using vs code and the command to open my NextJs project on it will be:

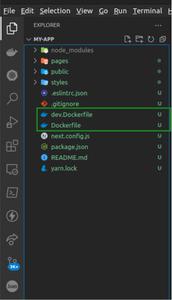
$ cd my-app && code .

**Step 3:**Create a Dockerfile in the root directory of your Next.js application.

A **Dockerfile** is a text document that contains all the commands a user could call on the command line to assemble an image. In this step, we will create two dockerfiles.

* Dockerfile: Dockerfile for production, alternatively you can name it prod.Dockerfile
* dev.Dockerfile: Dockerfile for development

Production means when the application is deployed for use and development means when the application is under development.



**This is the Dockerfile used in the production of the application:**

*# Dockerfile for production  
# Install dependencies only when needed  
FROM node:16-alpine AS deps*

*# Check https://github.com/nodejs/docker-node/tree/b4117f9333da4138b03a546ec926  
ef50a31506c3#nodealpine to understand why libc6-compat might be needed.  
RUN apk add –no-cache libc6-compat*

*WORKDIR /app*

*# Install dependencies based on the preferred package manager  
COPY package.json yarn.lock\* package-lock.json\* pnpm-lock.yaml\* ./  
RUN \  
if [ -f yarn.lock ]; then yarn –frozen-lockfile; \  
elif [ -f package-lock.json ]; then npm ci; \  
elif [ -f pnpm-lock.yaml ]; then yarn global add pnpm && pnpm i; \  
else echo “Lockfile not found.” && exit 1; \  
fi*

*# Rebuild the source code only when needed  
FROM node:16-alpine AS builder  
WORKDIR /app  
COPY –from=deps /app/node\_modules ./node\_modules  
COPY . .*

*# Next.js collects completely anonymous telemetry data about general usage.  
# Learn more here: https://nextjs.org/telemetry  
# Uncomment the following line in case you want to disable telemetry during the build.  
# ENV NEXT\_TELEMETRY\_DISABLED 1*

*RUN yarn build*

*# If using npm comment out above and use below instead  
# RUN npm run build*

*# Production image, copy all the files and run next  
FROM node:16-alpine AS runner  
WORKDIR /app*

*ENV NODE\_ENV production  
# Uncomment the following line in case you want to disable telemetry during runtime.  
# ENV NEXT\_TELEMETRY\_DISABLED 1*

*RUN addgroup –system –gid 1001 nodejs  
RUN adduser –system –uid 1001 nextjs*

*COPY –from=builder /app/public ./public*

*# Automatically leverage output traces to reduce image size  
# https://nextjs.org/docs/advanced-features/output-file-tracing  
COPY –from=builder –chown=nextjs:nodejs /app/.next/standalone ./  
COPY –from=builder –chown=nextjs:nodejs /app/.next/static ./.next/static*

*USER nextjs*

*EXPOSE 3000*

*ENV PORT 3000*

*CMD [“node”, “server.js”]*

You can create a file named “Dockerfile” in the root directory of your project and paste these instructions into it.

**This is the Dockerfile used in the development of your application:**

# dev.Dockerfile for development

FROM node:18-alpine

WORKDIR /app

# Install dependencies based on the preferred package manager

COPY package.json yarn.lock\* package-lock.json\* pnpm-lock.yaml\* ./

RUN \

if [ -f yarn.lock ]; then yarn --frozen-lockfile; \

elif [ -f package-lock.json ]; then npm ci; \

elif [ -f pnpm-lock.yaml ]; then yarn global add pnpm && pnpm i; \

else echo "Lockfile not found." && exit 1; \

fi

COPY . .

CMD yarn dev

**Step 4:**Altering next.config.js. To add support for Docker to an existing project, just copy the Dockerfile into the root of the project and add the following to the next.config.js file:

// next.config.js

module.exports = {

// ... rest of the configuration.

output: 'standalone',

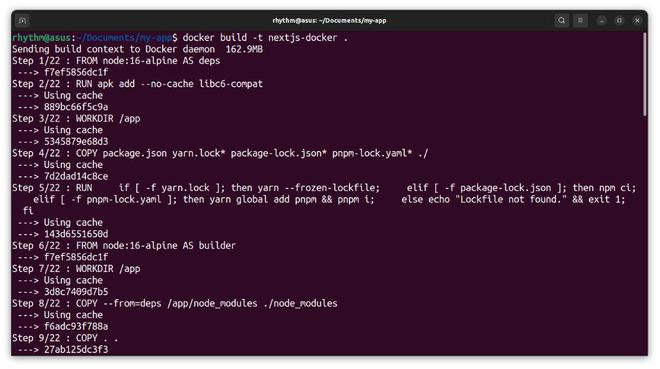
}

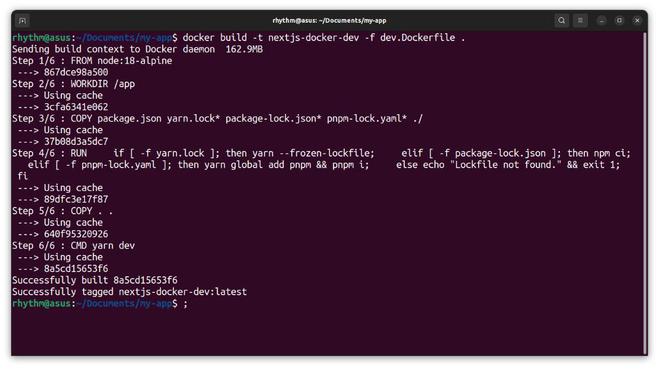
**Step 5:**Build The Dockerfile and dev.Dockerfile

$ docker build -t nextjs-docker .

$ docker build -t nextjs-docker-dev -f dev.Dockerfile .

It usually takes time to build the Image for the first time.





**Step 6:**Run your application. Based on the tags you gave to your Dockerfiles you can now run them withthe**docker run**command.

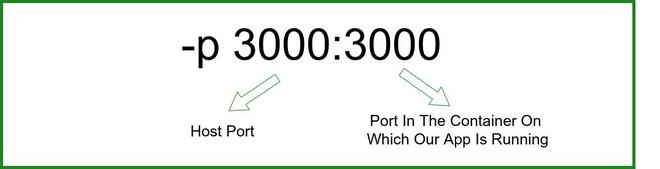
# For production

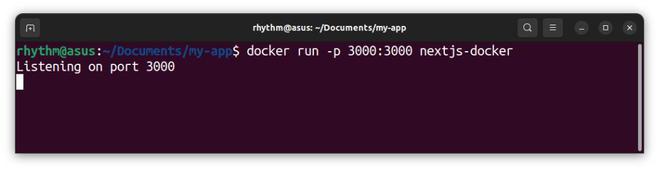
$ docker run -p 3000:3000 nextjs-docker

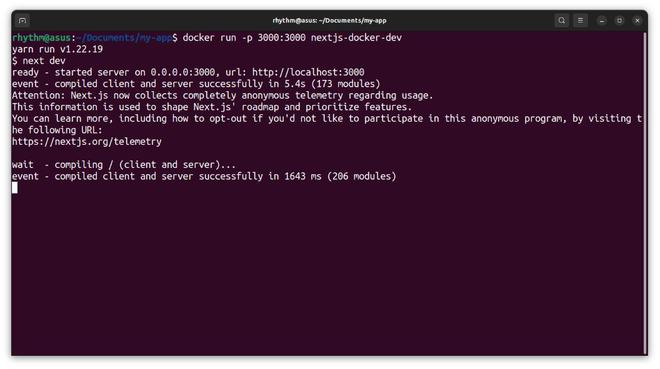
# For development, files wont be synced until step 8

$ docker run -p 3000:3000 nextjs-docker-dev

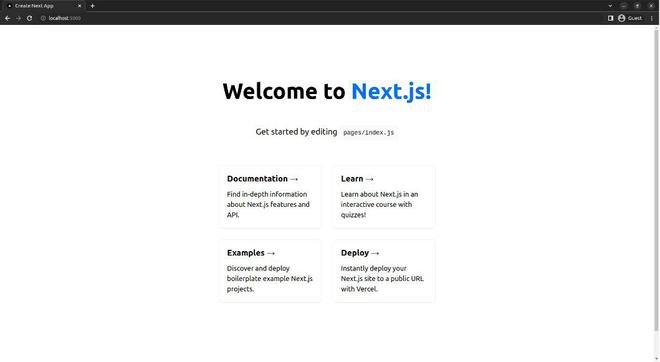
The -p flag exposes the container’s port to services outside docker.







**Step 7:** Verify if our application is running.

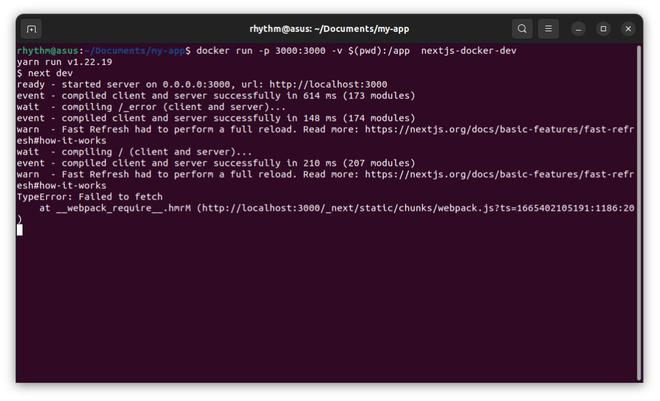


**Step 8:**Allow file change detection:

Everything looks great so far, but there is one small problem that we did not solve. As we know Images are read-only and any file change made after the files have been built will not reflect on the localhost. For this, we have to use a **bind mount**. With bind mounts, we control the exact mount point on the host. We can use this to persist data, but it’s often used to provide additional data into containers.

$ docker run -p 3000:3000 -v $(pwd):/app nextjs-docker-dev

-v $(pwd):/app specifies the mount point of the application so that file changes can be detected.



Try making some changes in your files and see if changes are being tracked. This was all about creating NextJs Docker Images. Read more about [NextJs](https://www.geeksforgeeks.org/tag/next-js/) on geeks for geeks.

# Data Storage in Docker

Last Updated : 30 Mar, 2023

Docker images are built-in form of layers and docker containers store all the data being used, on the container writable layer which is only persisted till the lifespan of the container i.e. it is no longer accessible once the container is removed. This also makes it difficult to get the data out of the container if it is required by some other processes.

To persist the data irrespective of the container’s lifecycle so that the files are available in the host filesystem even if the container is no longer available,

Docker provides two options:

1. **Docker volumes**
2. **Bind mounts**

Though we can also use***tmpfs mount***if you’re using Docker on Linux and ***named pipes*** if using Docker on Windows.

In this article, we’ll be covering the docker volumes and bind mounts with their differences in terms of use cases and effectiveness.

### ****1. Docker Volume****

Volumes are the directories or files that exist on the host filesystem and are mounted to the containers for persisting data generated or modified by them. They are stored in the part of the host filesystem **managed specifically by Docker and it should not be modified by non-Docker processes**. Volumes are the most preferred way to store container data as they provide efficient performance and are isolated from the other functionalities of the Docker host.

We can use the following command to create docker volume -:

docker volume create <volume\_name>

We can mount a volume to multiple containers simultaneously and Docker doesn’t remove them even if they’re not being used by any running container.

To remove a volume, we can use the following command -:

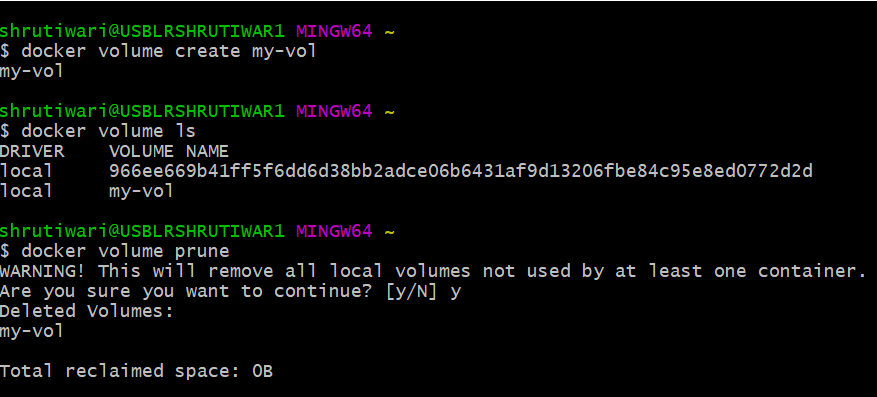
docker volume prune

In a Linux filesystem, we can find the available volumes at the following path **‘*/var/lib/docker/volumes/*‘**,

while in Windows, we can get the location of the volume by running the following command in file explorer -:

\\wsl$\docker-desktop-data\version-pack-data\community\docker\volumes

To learn more about volumes and how to mount them to a container click [here](https://www.geeksforgeeks.org/mounting-a-volume-inside-docker-container/).



*Commands to create volume*

### ****2.) Bind Mounts****

This is also a mechanism provided by Docker to store container data on localhost, but the directory or file mounted using bind mounts**can be accessed by non-Docker processes as well** and it relies on the host machine’s filesystem having a specific directory structure available because**it uses absolute path for binding**.

Bind mounts have limited functionality and can’t be managed directly through Docker CLI, thus making it less preferable in comparison to volumes. Moreover, **it allows the container to modify the *host* filesystem i.e. it can create, modify or delete important file directories which can impact security and non-Docker processes as well**.

### 3.) tmpfs mounts

When using Docker in Linux, you can also create storage volume for containers using *tmpfs*mount. But contrary to bind mounts and volumes, **this type of mount is temporary and persists on host memory.**

Once the container is stopped, the *tmpfs*mount is removed and files stored using it won’t be available anymore. This type of mount has very limited use and can only be used by Linux users. Moreover, **it does not allow sharing of mounted data among containers**.

### ****4.)  Named pipes (windows platform)****

### When to use Volumes

Following are some potential use cases for volumes -:

* Volumes can be used to share data among multiple containers in a secure manner without affecting the host filesystem.
* They provide convenient backup and data migration from one Docker host to another.
* We can easily manage volumes using Docker CLI and Docker APIs, which is a limitation in terms of bind mounts.
* The insignificance of host filesystem structure provides decoupling of Docker host configuration from container runtime.
* It provides volume drivers which help in storing data to remote hosts or cloud providers.
* Volumes are stored in Linux VM thus providing lower latency and higher throughput. They are highly performant on the Docker desktop, thus making them a better choice for write-intensive applications like data storage.

### ****When to use Bind Mounts****

Despite limited functionalities, we can use bind mounts in the following cases -:

* It can be used to provide shared configuration files between host and container. Ex- Docker mounts **‘*/etc/resolv.conf ‘*** to containers for DNS resolution.
* In case the host file system is guaranteed to remain consistent and the mounted volume is not accessed by non-Docker processes, we can use bind mounts for storing data

# How to use Docker Default Bridge Networking?

Last Updated : 17 Jun, 2024

Docker allows you to create dedicated channels between multiple Docker Containers to create a network of Containers that can share files and other resources. This is called Docker Networking. You can create Docker Networks with various kinds of Network Drivers which include Bridge drivers, McVLAN drivers, etc. By default, if you do not mention a driver while creating a network, it automatically chooses the default bridge driver. Bridge drivers are single-host networking drivers and hence their scope is limited to local.

**Table of Content**

* [Types of Docker Networks](https://www.geeksforgeeks.org/how-to-use-docker-default-bridge-networking/?ref=lbp#types-of-docker-networks)
* [Connect a Container To a User-Defined Bridge](https://www.geeksforgeeks.org/how-to-use-docker-default-bridge-networking/?ref=lbp#connect-a-container-to-a-userdefined-bridge)
* [To Create Custom Bridge Network](https://www.geeksforgeeks.org/how-to-use-docker-default-bridge-networking/?ref=lbp#to-create-custom-bridge-network)
* [How to create Custom Bridge Network? A Step-By-Step Guide](https://www.geeksforgeeks.org/how-to-use-docker-default-bridge-networking/?ref=lbp#how-to-create-custom-bridge-network-a-stepbystep-guide)
* [The Default Bridge Network](https://www.geeksforgeeks.org/how-to-use-docker-default-bridge-networking/?ref=lbp#the-default-bridge-network)
* [Connecting A Docker Container In Bridge Network](https://www.geeksforgeeks.org/how-to-use-docker-default-bridge-networking/?ref=lbp#connecting-a-docker-container-in-bridge-network)
* [Inspecting The Bridge Network](https://www.geeksforgeeks.org/how-to-use-docker-default-bridge-networking/?ref=lbp#inspecting-the-bridge-network)
* [Testing The Network Connectivity](https://www.geeksforgeeks.org/how-to-use-docker-default-bridge-networking/?ref=lbp#testing-the-network-connectivity)
* [Differences Between User-Defined Bridges And The Default Bridge](https://www.geeksforgeeks.org/how-to-use-docker-default-bridge-networking/?ref=lbp#differences-between-user-defined-bridges-and-the-default-bridge)
* [Difference between Docker Network Host and Bridge](https://www.geeksforgeeks.org/how-to-use-docker-default-bridge-networking/?ref=lbp#difference-between-docker-network-host-and-bridge)
* [Difference between Docker network bridge and Overlay](https://www.geeksforgeeks.org/how-to-use-docker-default-bridge-networking/?ref=lbp#difference-between-docker-network-bridge-and-overlay)

# Bridge Network Driver

The bridge is a default network where containers will be created by default if you are not mentioned any network while creating. The containers which are deployed in the same network can talk to each other the containers which are not in the same network can’t communicate with each other without proper mentions and permissions. While creating a docker network if you are not going to create or mention any network while creating a container. To list the networks in docker you can use the following command.

docker network ls

## Types of Docker Networks

There are three main default networks as mentioned following and also to know more about Docker networking refer to [Docker Networking.](https://www.geeksforgeeks.org/basics-of-docker-networking)

1. Bridge(default)
2. Host
3. None/Null

## **Connect a Container To a User-Defined Bridge**

If containers are created in a default bridge network. Communication will happen only with the[IP Address](https://www.geeksforgeeks.org/what-is-an-ip-address/)of the container. Communication will not happen using containerName(hostName). To Check Go inside[java web app container](https://www.geeksforgeeks.org/how-to-create-a-java-docker-container)and ping maven web app container using name & IP. When we ping using ip it will work it will not be able to communicate using the name.Developers should not code the connectivity based on the IP in case of containers. Since the IP address of containers will be dynamic.IP will keep changing.

## To Create Custom Bridge Network

To create a Coustm Bridge Network, we can use docker network command, the syntax of this as follows:

### ****Syntax****

docker network create -d <driver> <networkName>

## **Example1:**

## How to create Custom Bridge Network? A Step-By-Step Guide

The following are the steps that guides you in creating a custom bridge network:

**Step 1:** Ensure Docker is Running

* The following helps in checking the docker running status:

sudo systemctl status docker

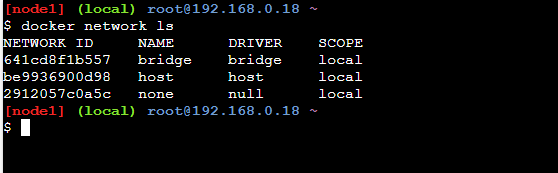
* The following command helps in enabling the docker service:

sudo systemctl enable docker --now

**Step 2:** Create Bridge Network

* Firstly check the existing networks list with the following command:

docker network ls



* Use the following command to create a new bridge network:

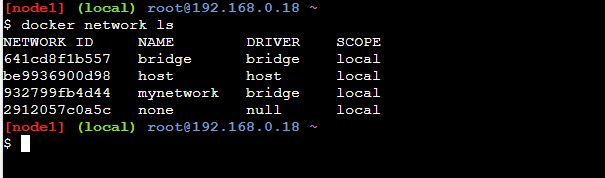
docker network create --driver bridge mynetwork



**Step 3:** Verify the Network Creation

* Use the following command to verify whether the network is successfully created or not:

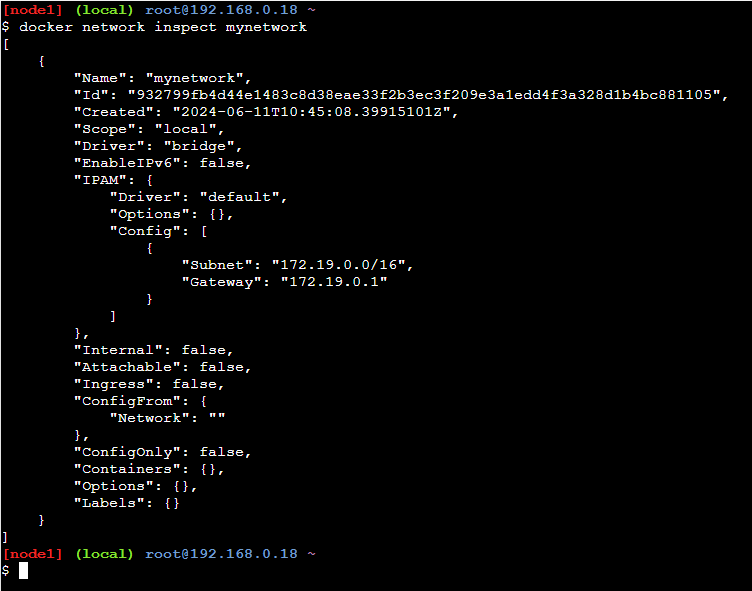
docker network ls



**Step 4:**Inspect the Bridge Network

* Inspect the created custom bridge networks by using the following command.

docker network inspect mynetwork

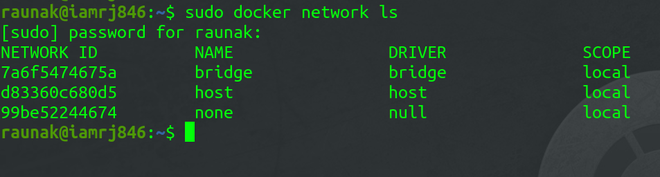


If containers are created in custom bridge network. Each container can access other using containerName/ContainerIP. Delete Containers which are running in default bridge or create container with different name.

## **The Default Bridge Network**

Every installation of Docker provides a pre-built default Bridge Network with a Bridge driver scoped locally. You can verify the same using the **network ls** to know more commands refer to command.[Docker – Instruction Commands.](https://www.geeksforgeeks.org/docker-instruction-commands)

sudo docker network ls



* Bridge Driver always provides single-host networking hence, the scope is local.

## **Connecting A Docker Container In Bridge Network**

Note that the Bridge Network we saw in the previous step is the default network for Docker Containers. If you don’t specify any other network, all new Containers will be joined to this default network. To connect an Ubuntu Container to the default bridge network, use this command.

sudo docker run -dt ubuntu

Connecting a Docker Container

## **Inspecting The Bridge Network**

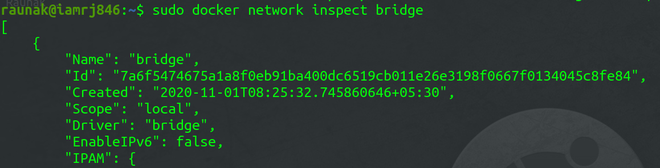
* After you have created the [Docker Container,](https://www.geeksforgeeks.org/containerization-using-docker/) check whether it is running or not.

sudo docker container ls

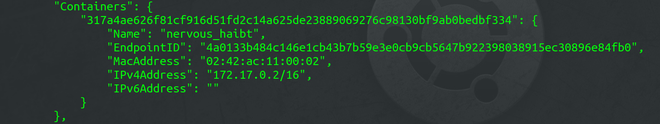
Inspecting the Bridge Network

* Since the Container is already running, we can now use the **network inspect**command to inspect the Docker default bridge network.

sudo docker network inspect bridge



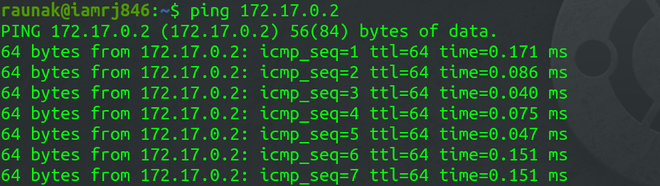
* You can see the details related to the Bridge Network in JSON format. You can also check the Containers associated with the network in the Container object.



## **Testing The Network Connectivity**

To test the network connectivity, note down the [IP address](https://www.geeksforgeeks.org/what-is-an-ip-address/) of the Container. In this example, the IP address is “172.17.0.2/16”.We will ping this address from the [Docker Host](https://www.geeksforgeeks.org/docker-containers-hosts/)to check the connectivity.

ping 172.17.0.2



It shows that the host is able to ping the Docker Container in the network.

## Differences Between User-Defined Bridges And The Default Bridge

| **Default Bridge Network** | **User Defined Bridges Network** |
| --- | --- |
| The default bridge network will it will acts as an basic network isolation for the containers which are deployed in this network. | user defined network will allows you to create a coustm network with more configuration policies. |
| The containers which are deployed in default bridge network will able to communicate with each other by **using the IP addresses.** | The containers which are deployed in the coustm bridge network will able to communicate with each other by using the **name of the containers.** |
| Communication with the host system is done by using the host IP addresses. | Communication with the host system is done by using the [host IP addresses](https://www.geeksforgeeks.org/what-is-network-id-and-host-id-in-ip-addresses/). |
| Containers use the embedded DNS to commutation to the internet by the DNS resolution. | Containers use the embedded [DNS](https://www.geeksforgeeks.org/domain-name-system-dns-in-application-layer/) to commutation to the internet by the DNS resolution. |

## Difference between Docker Network Host and Bridge

The following are the difference between docker network host and bridge:

| **Feature** | **Host Network** | **Bridge Network** |
| --- | --- | --- |
| **Network Isolation** | It shares the host’s network stack with providing no network isolation | it provides a private internal network, isolating containers from the host network |
| **Performance** | It offers better network performance due to direct use of the host’s network | It slightly lower performance due to the overhead of network bridging |
| **Use Case** | It is suitable for applications that needing direct access to host network, like monitoring tools | It is ideal for typical containerized applications needing isolation and controlled communication |
| **IP Addressing** | Containers use the host’s IP address and ports, leading to potential conflicts | Containers get their own IP addresses within the bridge network, avoiding conflicts with the host |

## Difference between Docker network bridge and Overlay

The following are the between docker network bridge and overlay:

| **Aspect** | **Bridge Network** | **Overlay Network** |
| --- | --- | --- |
| **Scope** | Limited to a single host | Spans multiple hosts |
| **Use Case** | Ideal for simple, local container communication | Suitable for distributed, multi-host environments |
| **Configuration** | Easy to set up with minimal configuration | Requires more complex setup, often involving a key-value store (like etcd) |
| **Performance** | It generally faster due to local scope | It is slightly slower due to cross-host communication overhead |

## Docker Bridge Network – FAQs

### What is a network bridge in Docker?

*A network bridge in Docker allows containers on the same host to communicate with each other through a virtual network interface.*

### What is the difference between Docker network host and bridge?

*The host network mode allows containers to share the host’s network stack, while the bridge network mode creates an isolated network namespace for containers on a single host.*

### What is the IP range of Docker bridge network?

*The default IP range of the Docker bridge network is `172.17.0.0/16`*

### What are the three types of network in Docker?

*Docker supports 1. bridge, 2. host, and 3.*

*overlay networks.*

### Can Docker networks talk to each other?

*By default, Docker networks are isolated, but you can connect containers across different networks using Docker’s networking features.*

# Docker – Docker Container for Node.js

Last Updated : 17 Jun, 2024

Node.js is an open-source, asynchronous event-driven JavaScript runtime that is used to run JavaScript applications. It is widely used for traditional websites and as API servers. At the same time, a Docker container is an isolated, deployable unit that packages an application along with its dependencies, making it highly scalable and maintainable. In this article, we will create a Docker container for node.js and run a simple express.js application on the container.

## What is Node.js?

[Node.js](https://www.geeksforgeeks.org/nodejs)is a server environment that is open-source and free to use. It provides the JavaScript runtime so that JavaScript applications can run on the server.  It also enables [JavaScript](https://www.geeksforgeeks.org/introduction-to-javascript/) code to run outside of a browser. It does a lot more, like generating dynamic page content, creating, saving, and and closing files on the server, and handling databases in the backend.

## **Step by Step to Dockerize NodeJS Application**

## Step 1: Dockerizing a Node.js web app

Let’s start with an effortless express application that prints “Hello World! This is Nodejs from a docker container” on visiting the root endpoint.

* To create a folder named **express\_app** and move inside the folder using the following commands.

mkdir express\_app  
cd express\_app

Now, create a file named **app.js** as follows.

* [**mkdir** stands for “make directory”.](https://www.geeksforgeeks.org/mkdir-command-in-linux-with-examples/)
* **express\_app** is the name of the directory you’re creating.
* [**cd**stands for “change directory”.](https://www.geeksforgeeks.org/cd-command-in-linux-with-examples/)
* **express\_app** is the directory you want to change into.

## **Step 2: Create the Node.js app**

Create a file that consists of all the files and dependencies required that describe your app and name it as a**“**[**package. Jason**](https://www.geeksforgeeks.org/node-js-package-json/)**“.**

{  
 "name": "docker-example",  
 "version": "1.0.0",  
 "description": "",  
 "main": "app.js",  
 "scripts": {  
 "start": "nodemon app.js",  
 "test": "echo \"Error: no test specified\" && exit 1"  
 },  
 "author": "",  
 "license": "ISC",  
 "dependencies": {  
 "express": "^4.17.1",  
 "nodemon": "^2.0.12"  
 }  
}

* Now, initialize the node project using the following command.

npm init

This will add the package.json file, which holds information about our projects like scripts, dependencies, and versions. It will ask for the package’s name, version, and many others (you can go with defaults by pressing ENTER).

* **npm**: This stands for Node Package Manager. It’s a tool that comes bundled with Node.js, and it’s used to manage packages (bits of code that you can use in your own projects) for Node.js. npm helps you install, uninstall, update, and manage these packages.
* **init**: This is short for “initialize”. When you run [npm init,](https://www.geeksforgeeks.org/node-js-npm-node-package-manager/) you’re asking npm to help you set up the initial configuration for your project. Essentially, you’re saying, “I want to start a new project. Please guide me through the setup process.”

Install the Express library and add it to the package.json file as a dependency.

npm install --save express

* **npm**: Stands for Node Package Manager. It’s a tool that helps developers manage and install packages (which are like plugins or libraries) for their Node.js applications.
* **install**: This is the command you’re giving npm. You’re telling it that you want to install something.
* **–save**: This part is saying that you want to save the package you’re installing as a dependency in your project’s package.json file. The package.json file keeps track of all the dependencies your project needs to run. Saving the package as a dependency means that if someone else wants to run your project, they can just look at the package.json file, see what packages are needed, and install them easily.
* **express**: This is the name of the package you want to install. In this case, it’s a popular package for creating web servers and building web applications with Node.js. When you install it, npm will fetch all the necessary files and put them in a folder called node\_modules in your project directory. This folder contains all the code for the packages you’ve installed.

Install a tool called nodemon that automatically restarts the node application when it detects any changes.

npm install --save nodemon

We are explicitly adding these dependencies to our package.json file to download them when we run this application inside a docker container.

* Add a script to the scripts part of the [package.json](https://www.geeksforgeeks.org/node-js-package-json) file to run the application with nodemon. The file contents will be as follows:

## Step 3: Express.js

Express is a framework for Node.js that helps developers organize their web applications into [MVC architecture](https://www.geeksforgeeks.org/mvc-framework-introduction/) on the server side. It makes web applications fast and easy as compared to developing an application using only Node.js.

For example, A node.js application might need [MongoDB](https://www.geeksforgeeks.org/mongodb-an-introduction) in the backend, and Express is the framework that helps manage everything like routes, requests, and views.

Then, create a server.js file that defines a web app using the Express.js framework:

// import and create an express app  
const express = require('express');  
const app = express()  
  
// message as response  
msg = "Hello world! this is nodejs in a docker container.."  
// create an end point of the api  
app.get('/', (req, res) => res.send(msg));  
  
// now run the application and start listening  
// on port 3000  
app.listen(3000, () => {  
 console.log("app running on port 3000...");  
})

At this stage, we can run our application on our local system using the following command:

npm run start

But actually, we want to dockerize this application. For that, we need to create an image by providing information like which runtime we need, the port the app will use, and the files needed that are available on our local system.

* **npm**: npm is a command-line tool that comes bundled with Node.js. It’s used for managing Node.js packages and dependencies, and also for running scripts defined in the package.json file.
* **run**: This tells npm that you want to execute a script.
* **start**: This is the name of the script you want to run. It’s a special name recognized by npm. You can customize it, but "start" is a convention for running the main entry point of your application.
* Create a Dockerfile that contains all the information about the image that will run the application. The docker software understands this special file, and it is used to build an image.

## **Step 4:**Create a Dockerfile in your Node.js app project

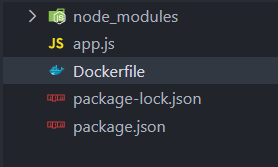
First, create a file name as[Dockerfile](https://www.geeksforgeeks.org/docker-concept-of-dockerfile).

FROM node:latest  
WORKDIR /app  
COPY package.json /app  
RUN npm install  
COPY . /app  
CMD ["npm", "start"]

### ****Explanation of Dockerfile****

1. The FROM takes the name of the base image to use optionally with its version.
2. WORKDIR states the directory that holds the files of the application in the container.
3. COPY command copies the package.json file to the app directory.
4. The RUN command runs the provided command to install all the dependencies mentioned in the package.json file.
5. Then COPY is used to copy the rest of the files to the app directory in the container.
6. Finally, we provide the script to run the application.

* The folder structure after creating all the required files is as follows:



* Finally, use this command to build the image we will run in our docker container.

## Step 5: Building Your Image

After creating a Dockerfile build an image using the help Dockerfile by using the following command.

docker build -t docker-container-nodejs .

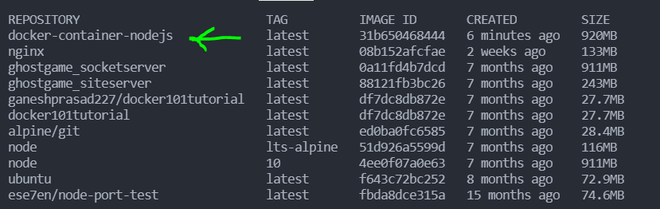
* **docker build:** This is the command used to build a Docker image. It tells Docker to build an image based on the instructions provided in the Dockerfile.
* **-t docker-container-nodejs:** The -t flag is short for “tag,” and it is used to specify a name and optionally a tag for the image being built. In this case, we’re giving the image the name “docker-container-nodejs”. The name is typically in the format <repository>/<image\_name>, where docker-container-nodejs is the image name.
* **.:** The dot . at the end of the command represents the build context. It tells Docker to look for the Dockerfile and any other necessary files in the current directory. The build context is where Docker looks for files referenced in the Dockerfile, such as source code, dependencies, and configuration files.

The command uses the flag -t to specify the name of the image, and then we have to give the address where. Then, our Dockerfile is situated; since we are in the directory while running the commands, we can use the period, which stands for the current directory.

* Confirm that the image has been created.

docker images

The output will be a list of images in your system. It should contain our recently created image with the name we provided with the -t flag.



* To run the docker container with this image, use the following command.

## Step 6: Run the image

Run the image as a container and deploy the application in the form of containers.

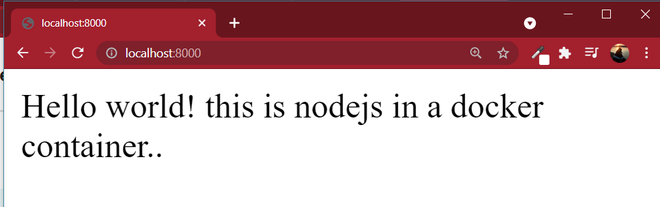
docker run -d -p 8000:3000 -v address\_to\_app\_locally:/app docker-container-nodejs

The above command runs a docker container. The flag -p is used to map the local port 8000 to the container’s port 3000 where our application is running. The -v flag is used to mount our application files into the app directory of the container. It also needs the image name that we want to run in our container, which is, in this case, docker-container-nodejs that.

* **docker run**: This command is used to create and start a new Docker container from an image.
* **-d**: This flag stands for “detached mode”. It runs the container in the background, so your terminal is free for other commands.
* **-p 8000:3000**: This option maps a port on your host machine to a port inside the Docker container.
  + **8000**: The port on your host machine (i.e., your computer).
  + **3000**: The port inside the Docker container where the Node.js application is running.
  + This means that when you access http://localhost:8000 on your host machine, it will be forwarded to http://localhost:3000 inside the container.
* **-v address\_to\_app\_locally:/app**: This flag mounts a volume, linking a directory on your host machine to a directory inside the container.
  + **address\_to\_app\_locally**: The path to your application’s directory on your host machine.
  + **/app**: The directory inside the Docker container where the application will be available.
  + This allows you to develop your application locally and see changes inside the container without rebuilding it.
* **docker-container-nodejs**: This is the name of the Docker image from which the container is created. Make sure you have built or pulled this image before running the command.

## Step 7: Test the Application

 Visit this address [**localhost**](https://www.geeksforgeeks.org/what-is-local-host)**:8000**, and our express application will return the following response.



*Express app on a docker container*

## Using Docker Compose for node

version: '3.8'  
  
services:  
 app:  
 build: .  
 ports:  
 - "8000:3000"  
 volumes:  
 - .:/app  
 command: npm start

* **version: '3.8'**: Specifies the version of the Docker Compose file format. Version 3.8 is commonly used and supports the latest features.
* **services**: Defines the services that make up your application. Here, we have one service named app.
* **app**: The name of the service. This service represents your Node.js application.
* **build: .**: Instructs [Docker Compose](https://www.geeksforgeeks.org/installation-of-docker-compose/) to build an image from the Dockerfile located in the current directory (.).
* **ports**: Maps ports between the host machine and the Docker container.
  + **"8000:3000"**: Maps port 8000 on your host machine to port 3000 inside the Docker container. When you access http://localhost:8000 on your host machine, it forwards the request to port 3000 in the container where the Node.js app is running.
* **volumes**: Mounts a directory from the host machine into the Docker container.
  + **. : /app**: Mounts the current directory on your host machine (.) to the /app directory inside the container. This allows you to edit your code on your host machine, and the changes will be reflected inside the container.
* **command**: Specifies the command to run within the container.
  + **npm start**: Runs npm start, which typically starts your Node.js application.

## **Docker Node best practices**

* **Use Official Node Images**: Start with a [lightweight base image](https://www.geeksforgeeks.org/how-to-optimize-docker-image/) like node:alpine for smaller image sizes.
* **Leverage Multi-Stage Builds**: Use multi-stage builds to reduce the final image size by only including necessary files.
* **Cache Dependencies**: Copy package.json and package-lock.json first and run npm install before copying other files to leverage [Docker’s layer caching.](https://www.geeksforgeeks.org/introduction-to-efficient-docker-caching-strategies/)
* **Set Working Directory**: Use WORKDIR to set a working directory inside the container.
* **Use .dockerignore**: Create a[.dockerignore file](https://www.geeksforgeeks.org/how-to-use-a-dockerignore-file/)to exclude unnecessary files and directories from the build context.
* **Run as Non-Root User**: Add a non-root user inside the Dockerfile and switch to it for better security.
* **Optimize CMD**: Use CMD ["node", "your-app.js"] to define the default command for the container.

### How to build a node docker image?

*To build a Node.js Docker image, create a Dockerfile with the necessary instructions, then run:*

*docker build -t your-image-name .*

### What is node in docker?

*Node in Docker refers to running a Node.js application inside a Docker container, which provides an isolated and consistent environment for development, testing, and deployment. This setup ensures the application runs the same way across different environments.*

### What is a node image?

*A Node image is a pre-configured Docker image that includes the Node.js runtime and libraries, allowing developers to run Node.js applications within a Docker container. It provides an isolated environment for application development and deployment.*

### How big is a node docker image?

*The size of a Node.js Docker image varies by version and type (e.g., node:alpine is lighter). For example, node:alpine is around 50 MB, while the full node image can be around 900 MB.*

### How to run node image from docker command?

*To run a Node.js image from a Docker command, use:*

*docker run -it --rm -v "$(pwd)":/app -w /app node:latest node your-app.js*

*This runs your Node.js application (your-app.js) in an interactive terminal, removes the container after it exits, and mounts the current directory into /app inside the container.*

# How to Install Linux Packages Inside a Docker Container?

Last Updated : 06 Oct, 2021

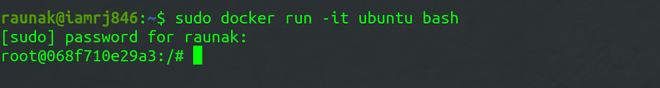
Once you understand how to pull base Docker Images from the Docker registry, you can now simply pull OS distributions such as Ubuntu, CentOS, etc directly from the Docker hub. However, the OS Image that you have pulled simply contains a raw file system without any packages installed inside it. When you work on projects inside Docker Containers, you will definitely need some libraries and packages either right from the beginning of the development phase or on the go.

In this article, we will see how to install packages and libraries inside Docker Containers using 2 different methods. We will also try to install Firefox and vim text editor inside Ubuntu Docker Container.

### ****Method 1:**** ****Using Command Line Interface****

**Step 1:**Open the terminal of your local system and run the Ubuntu Docker Image from the Docker Registry. If your system has no previous pulls, it will start pulling from the registry.

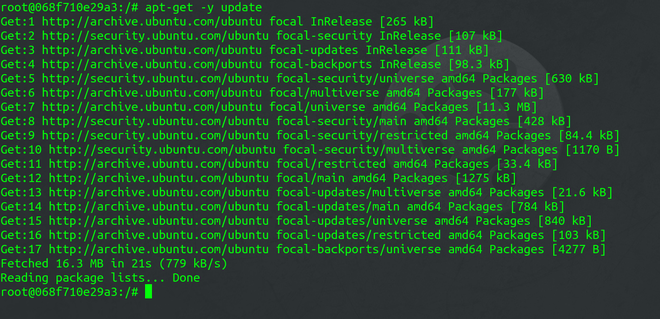
sudo docker run -it ubuntu bash



*Running Ubuntu Container*

**Step 2:** Now, you have opened the bash of your Ubuntu Docker Container. To install any packages, you first need to update the OS.

apt-get -y update

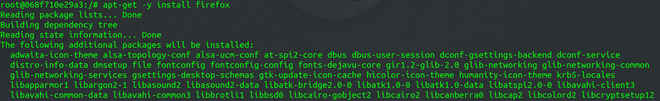


*Updating the Container*

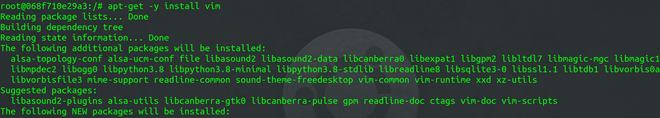
**Step 3:** After you have updated the Docker Container, you can now install the Firefox and Vim packages inside it.

apt-get -y install firefox

apt-get -y install vim



*Installing Firefox*



*Installing Vim*

You can now easily use these packages through the bash itself.

### ****Method 2: Using Dockerfile****

You can also directly specify the packages you need to install in the *dockerfile* using the **RUN** instruction. This method is preferable over the **CLI** method because building a *dockerfile* is very essential if you are working on projects inside Docker. It gives better version control and provides a blueprint of the entire Docker Image.

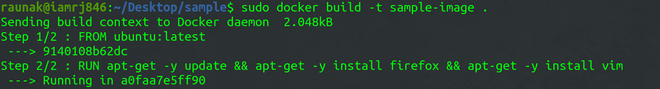
**Step 1:** Write your instructions inside a *dockerfile*.

FROM ubuntu:latest

RUN apt-get -y update && apt-get -y install firefox && apt-get -y install vim

**Step 2:** Build the Image using the Docker Build command.

sudo docker build -t sample-image .



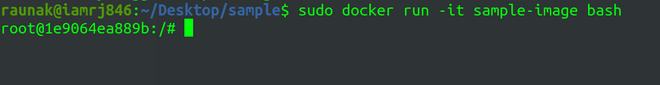
*Building the Image*

**Step 3:** Run the container and verify the installed packages

sudo docker run -it sample-image bash

vim

firefox



*Running the Container*



*Vim Text Editor running inside Container*

# Introduction to Docker Swarm Mode

Last Updated : 10 Jun, 2024

Docker swarm is a container orchestration tool. Swarm Mode in Docker was introduced in version 1.12 which enables the ability to deploy multiple containers on multiple Docker hosts. For this Docker uses an overlay network for the service discovery and with a built-in load balancer for scaling the services. One of the key advantages of docker swarm is that the configuration of the docker services such as volume and network can be modified without the need to manually restart the service Docker will update the configuration, stop the service tasks with the out-of-date configuration, and create new ones matching the desired configuration.

**Table of Content**

* [What is Docker and Docker Container?](https://www.geeksforgeeks.org/introduction-to-docker-swarm-mode/#what-is-docker-and-docker-container)
* [What is Docker Swarm?](https://www.geeksforgeeks.org/introduction-to-docker-swarm-mode/#what-is-docker-swarm)
* [What are the Task and Service in Docker Swarm?](https://www.geeksforgeeks.org/introduction-to-docker-swarm-mode/#what-are-the-task-and-service-in-docker-swarm)
* [How Does Docker Swarm Work?](https://www.geeksforgeeks.org/introduction-to-docker-swarm-mode/#how-does-docker-swarm-work)
* [Docker Swarm Init](https://www.geeksforgeeks.org/introduction-to-docker-swarm-mode/#docker-swarm-init)
* [Docker Swarm Architecture](https://www.geeksforgeeks.org/introduction-to-docker-swarm-mode/#docker-swarm-architecture)
* [Features of Docker Swarm](https://www.geeksforgeeks.org/introduction-to-docker-swarm-mode/#features-of-docker-swarm)
* [What is Docker Swarm Used For?](https://www.geeksforgeeks.org/introduction-to-docker-swarm-mode/#what-is-docker-swarm-used-for)
* [Different Modes of Docker Swarm](https://www.geeksforgeeks.org/introduction-to-docker-swarm-mode/#different-modes-of-docker-swarm)
* [How to Install and Setup Docker Swarm on AWS? A Step-By-Step Guide](https://www.geeksforgeeks.org/introduction-to-docker-swarm-mode/#how-to-install-and-setup-docker-swarm-on-aws-a-stepbystep-guide)
* [How to Setup Docker Swarm Mode?](https://www.geeksforgeeks.org/introduction-to-docker-swarm-mode/#how-to-setup-docker-swarm-mode)
* [How to Create a Task in Docker Swarm Cluster?](https://www.geeksforgeeks.org/introduction-to-docker-swarm-mode/#how-to-create-a-task-in-docker-swarm-cluster)
* [Getting Started With Swarm Mode](https://www.geeksforgeeks.org/introduction-to-docker-swarm-mode/#getting-started-with-swarm-mode)
* [What is Stack in Docker Swarm?](https://www.geeksforgeeks.org/introduction-to-docker-swarm-mode/#what-is-stack-in-docker-swarm)
* [Docker Swarm Mode Key Concepts](https://www.geeksforgeeks.org/introduction-to-docker-swarm-mode/#docker-swarm-mode-key-concepts)
* [Docker Swarm Filters](https://www.geeksforgeeks.org/introduction-to-docker-swarm-mode/#docker-swarm-filters)
* [Docker Swarm Mode CLI Commands](https://www.geeksforgeeks.org/introduction-to-docker-swarm-mode/#docker-swarm-mode-cli-commands)
* [Docker Container vs Docker Swarm](https://www.geeksforgeeks.org/introduction-to-docker-swarm-mode/#docker-container-vs-docker-swarm)
* [Docker Swarm vs Kubernetes](https://www.geeksforgeeks.org/introduction-to-docker-swarm-mode/#docker-swarm-vs-kubernetes)
* [Docker Service vs Docker Task](https://www.geeksforgeeks.org/introduction-to-docker-swarm-mode/#docker-service-vs-docker-task)
* [Benefits of Docker Swarm](https://www.geeksforgeeks.org/introduction-to-docker-swarm-mode/#benefits-of-docker-swarm)

## **What is Docker image and Docker Container?**

Docker is an open-source container platform that helps in packing an appliation with all its dependencies in on one bundle i.e., Docker Image. On using this docker image assigning the resources such as RAM, CPU executing it as process known as Docker Container. Docker platform make the application containerized and portable to the all systems that supports docker.

## What is Docker Swarm?

Docker Swarm is a native clustering and orchestrating tool that helps in managing the a docker engines. In this the group of docker engines turned into as single virtual docker host. It facilitates the users to deploy, manage and scale the applications seamlessly across multiple docker nodes. It comes with providing the features such as service discovery, load balancing, scaling, and rolling updates. It provides an easy and integrated way to manage the containerized applications in a cluster.

To know more about Docker Swarm refer to this – [Article](https://www.geeksforgeeks.org/introduction-to-docker-swarm-mode/)

## **What are the Task and Service in Docker Swarm?**

In Docker Swarm, a **Service** is a higher-level abstraction used to define how containers should be deployed, managed, and scaled across a swarm of Docker nodes. It includes specifications like the number of replicas, networking, and load balancing.

A **Task** is an individual instance of a container that is created and managed by a service. Each task represents a single unit of work and is scheduled to run on one of the nodes in the swarm. Tasks are the actual running containers that fulfill the requirements specified by the service

## **How Does Docker Swarm Work?**

When you want to deploy a container in the swarm first, you have to launch services. Service consists of multiple containers of the same image. These services are deployed inside a node so to deploy a swarm at least one node has to be deployed. As you see below diagram the manager node is responsible for the allocation of the task, dispatch the tasks, and schedule the tasks. [API](https://www.geeksforgeeks.org/what-is-an-api/)in the manager is the medium between the manager node and the worker node to communicate with each other by using the [HTTP protocol](https://www.geeksforgeeks.org/types-of-internet-protocols/). The service of one cluster can be used by the other. All the execution of the task is performed by the worker node.

## **Docker Swarm Init**

To initialize the docker swarm cluster we use the command called**“docker swarm init”.**For converting the docker engine to the swarm manager we will use this command after converting into the swarm mode then you will able the manager and the worker nodes then the swarm will distribute the work across them.

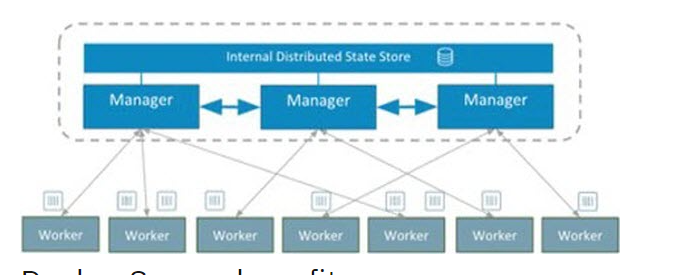
### After Initializing The Swarm Mode

* After initializing the docker swarm init it will bring the swarm into existence.
* After bringing the swarm into existence know it will turn the current node into a manager node. And also it will generate one token.
* The token that is generated will be used for the further joining of the worker nodes and master nodes.

## **Docker Swarm Architecture**

There are two types of nodes in Docker Swarm:

1. **Manager node:** Carries out and oversees cluster-level duties.
2. **Worker node:**Receives and completes the tasks set by the manager node.



A single manager node can be created but the worker node can not be created without a manager node. The ideal number for the count of the manager node is seven. Increasing the number of the manager node does not mean that the [scalability will increase](https://www.geeksforgeeks.org/overview-of-scaling-vertical-and-horizontal-scaling/).

## **Features of Docker Swarm**

The following are the features of Docker Swarm:

1. **Cluster management:** To create Swarm you can use the [Docker engine CLI](https://www.geeksforgeeks.org/tips-to-manage-docker-containers-using-cli/) where you can deploy the applications. Additional orchestration software is not required to manage a swarm.
2. **Multi-host networking:**Swarm can contain multiple overlay networks so while deploying the service you can specify the network on which you want to deploy your service. The swarm manager automatically assigns addresses to the containers on the overlay network when it initializes or updates the application.
3. **Load balancing:** While deploying any service on a particular port the swarm automatically balances the load of these ports.
4. **Scaling:** When you scale up or down, the swarm manager automatically adapts by adding or removing tasks to maintain the desired state.

## **What is Docker Swarm Used For?**

Docker swarm is a container orchestration tool that is used to Docker containers and scale them. Instead of a single host with the help of Docker Swarm, we can manage multiple nodes which are called clusters where we can deploy and maintain our containers in multiple hosts.

1. Docker Swarm assures that our application is always available even if one of the nodes fails by creating the container in another node which is available
2. Based on the incoming traffic we can scale the containers up and down by adding to the multiple nodes.
3. If there are multiple containers the incoming load will be balanced automatically by the Docker Swarm.
4. Docker Swarm has a number of security features, including traffic encryption between nodes and mutual TLS authentication.
5. Docker Swarm will automatically take care of failed containers and nodes. By this, we can maintain high availability.

Simply Docker Swarm is mainly used to deploy, scale, and manage the containers and nodes which are available in the cluster.

## **Different Modes of Docker Swarm**

Docker Swarm mainly consists of two modes they are:

1. **Global Mode:**In this mode, Docker Swarm will maintain containers in all the slave nodes and master nodes. It will maintain the replicas of containers in all the nodes which are available in the cluster.
2. **Replicated Mode:**In this mode, Docker Swarm will deploy the containers based on the no.of replicas required for you. If you required 3 replicas it will deploy the containers based on the node availability.

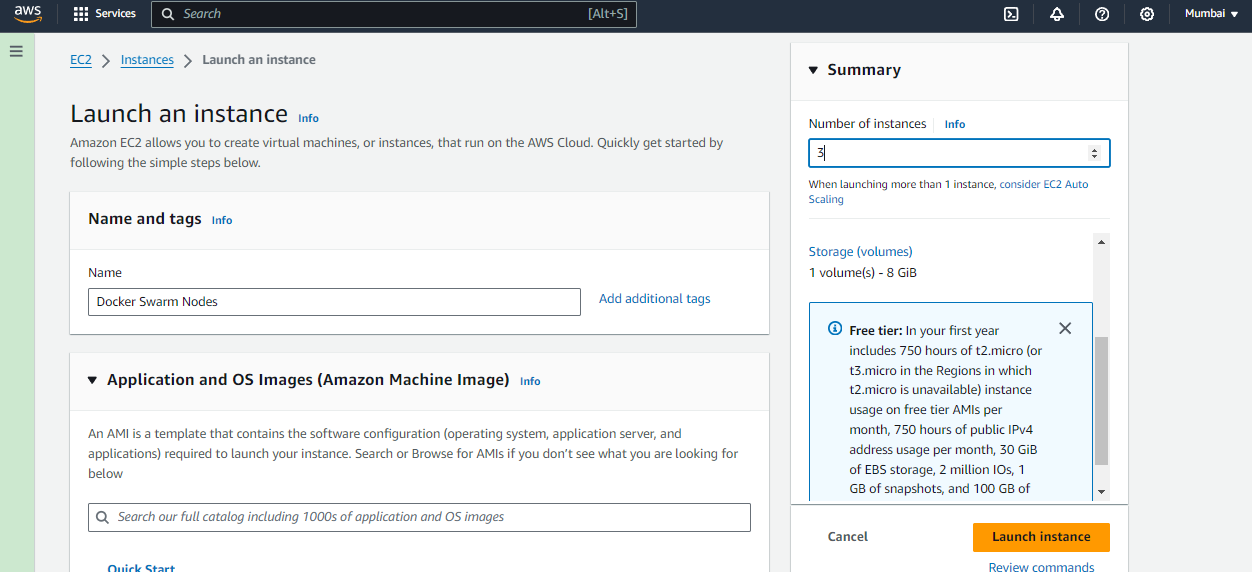
## **How to Install and Setup Docker Swarm on AWS? A Step-By-Step Guide**

**Step 1:** Login to AWS Console

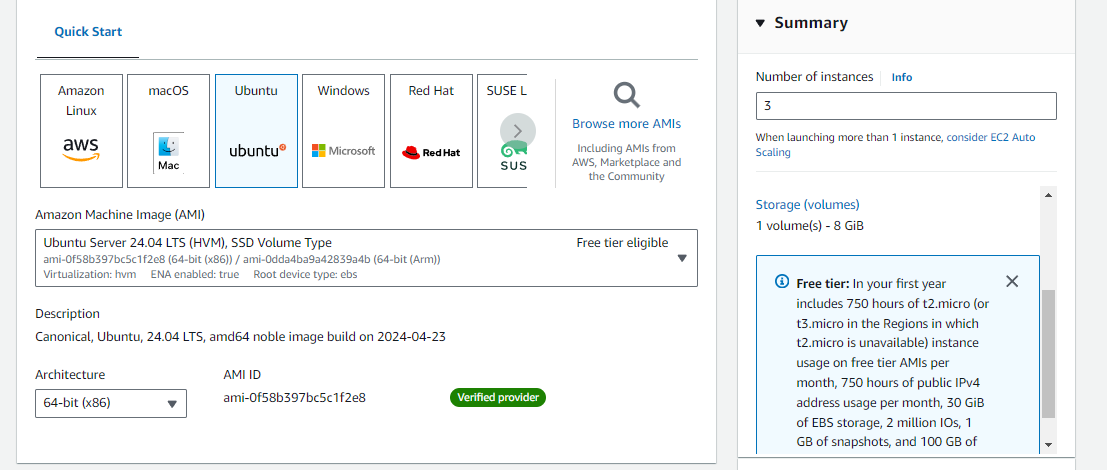
* Navigate to [AWS Account](https://www.geeksforgeeks.org/amazon-web-services-setting-up-an-aws-account/)with your credentials.
* Then navigate[EC2](https://www.geeksforgeeks.org/what-is-elastic-compute-cloud-ec2/) Dashboard

**Step 2:**Launch the EC2 Instances

* Launch the 3 EC2 instances with ubuntu AMI (Here we taking 1 docker manager and 2 swarm nodes ).

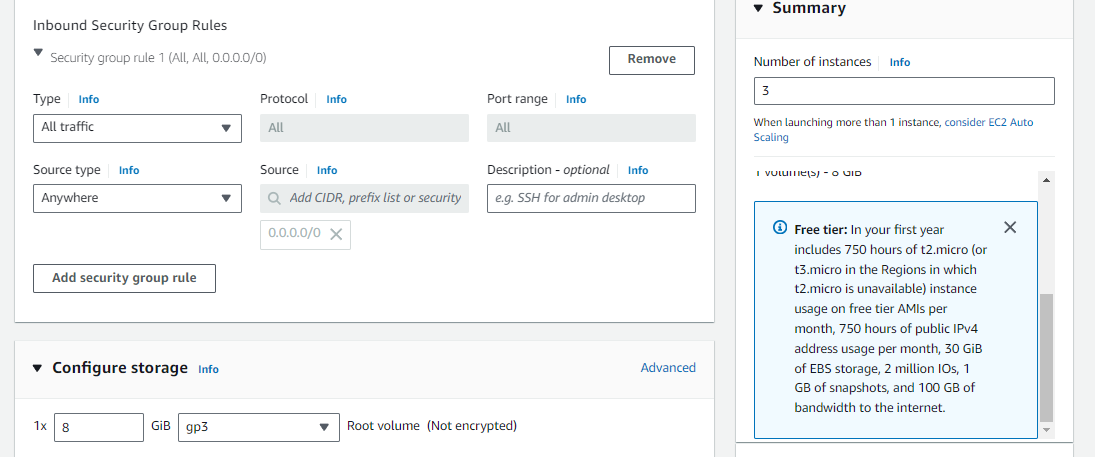


* Choosing AMI as ubuntu and the instance type as “t2.micro”

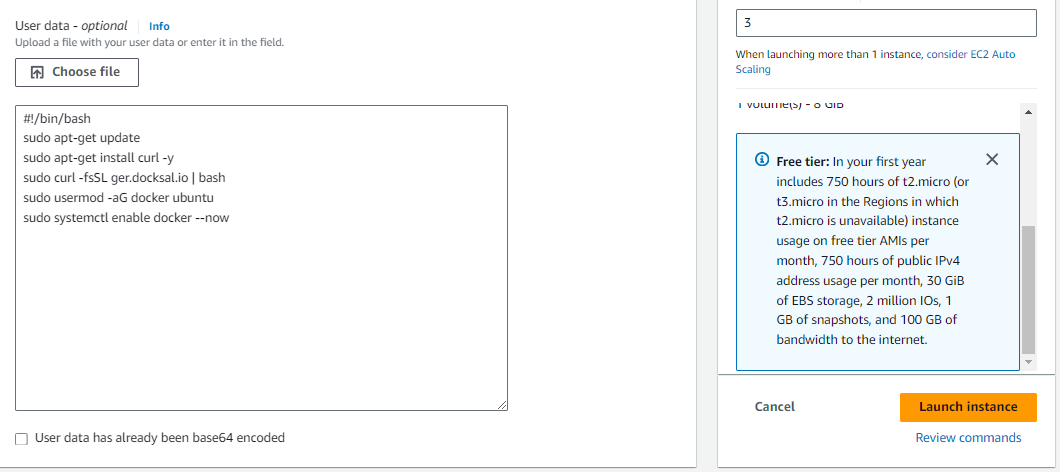


* Choose the key pair option as No key pair, because here we demonstrating through web console.

**Step 3:**Specify the inbound Security rules



**Step 4:**Specify the Metadata to instances to automate the docker setup process to all the nodes at once.

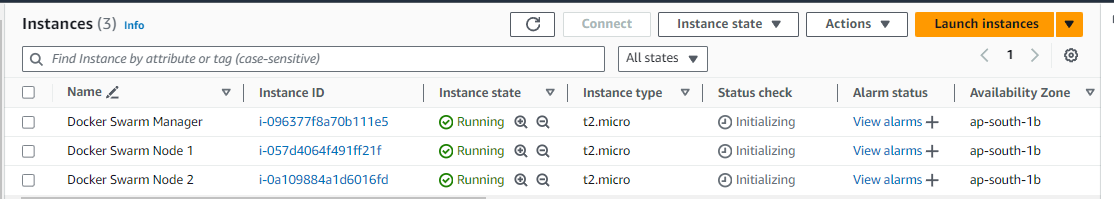


( or )

* All specify this below code in this metadata.

#!/bin/bash  
  
# Update package lists and install dependencies  
sudo apt-get update -y && sudo apt-get install -y \  
 apt-transport-https \  
 ca-certificates \  
 curl \  
 software-properties-common  
  
# Add Docker’s official GPG key  
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -  
  
# Set up the stable repository for Docker  
sudo add-apt-repository -y \  
 "deb [arch=amd64] https://download.docker.com/linux/ubuntu \  
 $(lsb\_release -cs) \  
 stable"  
  
# Update the package database with Docker packages from the newly added repo  
sudo apt-get update -y  
  
# Install Docker  
sudo apt-get install -y docker-ce  
  
# Start Docker service  
sudo systemctl start docker  
  
# Enable Docker to start on boot  
sudo systemctl enable docker  
  
# Install Docksal  
sudo curl -fsSL ger.docksal.io | bash  
  
# Add the current user to the docker group  
sudo usermod -aG docker "${USER}"

**Step 5:** Specify the names to the instances after once they created as shown in the below figure:

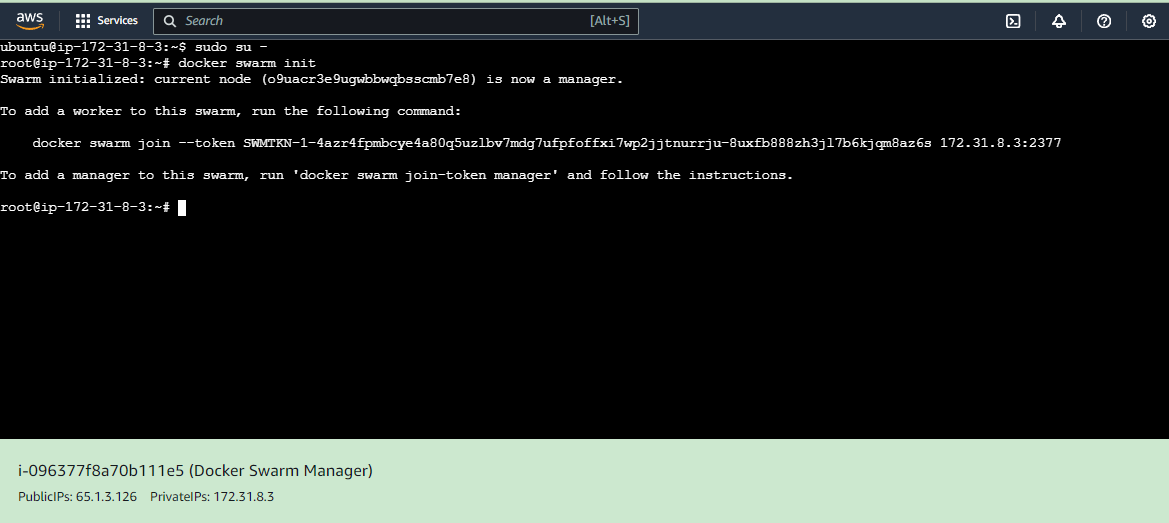


## How to Setup Docker Swarm Mode?

**Step 6:**StartDocker Swarm

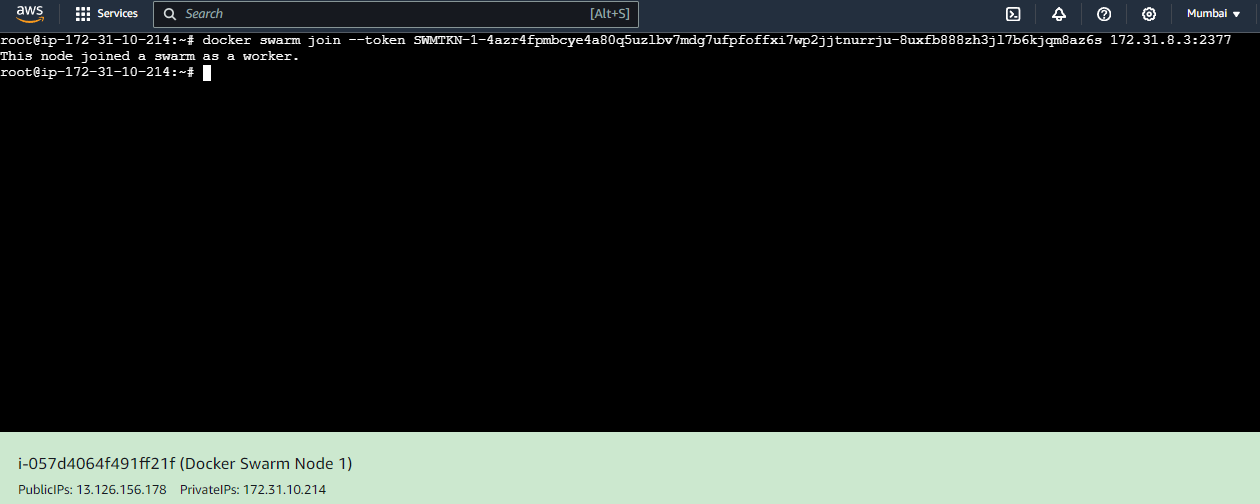
* The following command helps to switch to the root user
* Run the following command to get the token for docker swarm joining.

docker swarm init

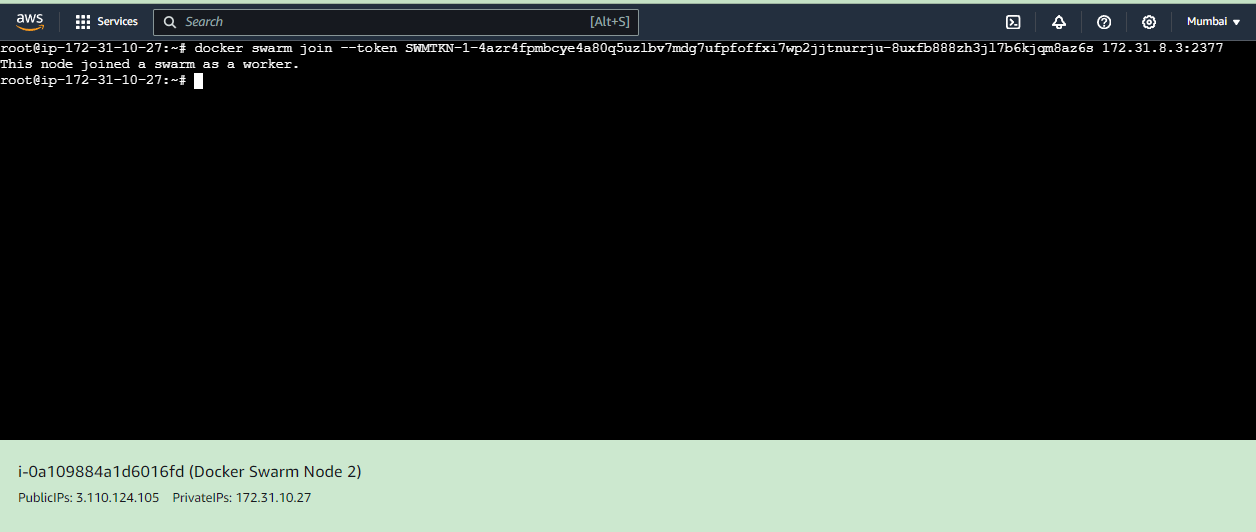


**Step 7:** Joining Docker Swarm Nodes

* Copy the token from the above step and run in the **docker swarm node 1**to join into the docker swarm cluster.



* Copy the token from the above step and run in the **docker swarm node 2**to join into the docker swarm cluster.

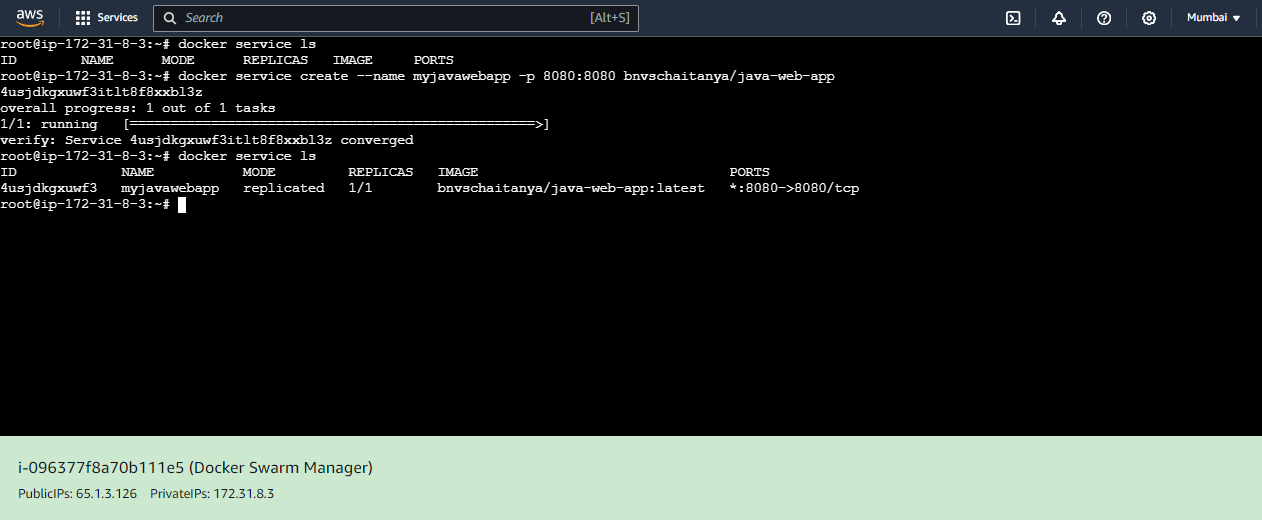


## How to Create a Task in Docker Swarm Cluster?

**Step 8**: Create a Task in Docker Swarm

Run the following command in docker swarm manager to create a task in the swarm cluster:

docker service create --name myjavawebapp -p 8080:8080 bnvschaitanya/java-web-app



**Step 9:**Create a Replica to Task

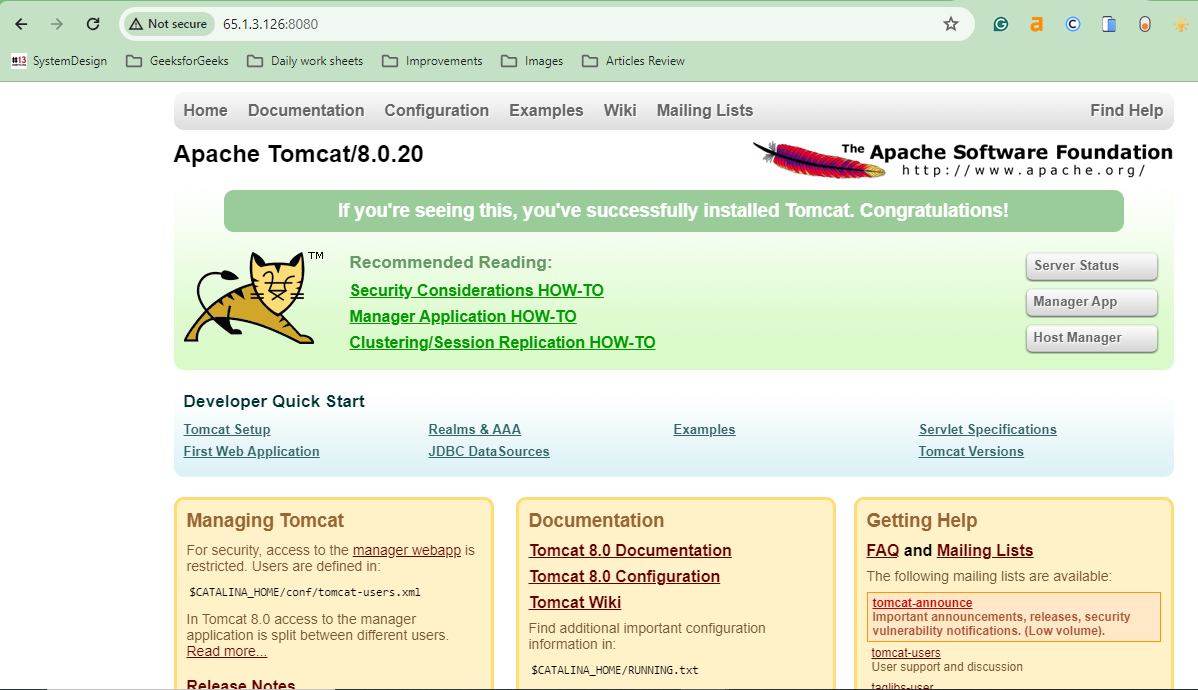
* Scale the created service from default 1 to 4 with the following command:

docker service scale myjavawebapp=4



**Step 10:** Access the application through browser using any of your cluster instance ip with 8080 port as follows:

http://65.1.3.176:8080



## **Getting Started With Swarm Mode**

Follow the steps mentioned below to get familiar with the docker swarm mode.

**Step 1:** In the first step we need to initialize the docker engines of swarm mode.

**Step 2:** After initializing the swarm mode know add the nodes into the docker swarm cluster. When you intilize the swarm mode it will generates two tokens one is for the manger node and another is for the worker node by using following command you can join the nodes according to the requirement.

docker swarm init <Token>

**Step 3:** Know you can start deploying the you application in the form of containers in docker swarm. Docker swarm will take care of deployment of application an scaling of the application across the worker nodes.

**Step 4:**WIth the help od docker swarm cli you can manage the swarm like adding or removing the worker nodes scaling the services and inspecting the swarm state.

## **What is Stack in Docker Swarm?**

A stack is nothing but a collection of one or more services deployed as a single unit. The stack is deployed by using compose file in which you can mention the service of the stack and all the required configurations to deploy the stack.

With the help of stack, it is very easy to deploy and maintain complex containers like multi-containers in the Docker swarm. We can deploy with the help of a single file called **Docker Compose. yaml**we can define all the necessary configurations. You can be assured that they deployed and scaled together.

If we deploy the new version of the application and the end users find any bugs while using it you can roll back to the previous version of the application by using Docker Swarm.

**Note:**Stack is mainly used to deploy the multi-container application with ease.

## Docker Swarm Mode Key Concepts

The following are the docker swarm mode key concepts:

* **Node:**A Node is an instance of a Docker engine that connects to the Swarm. You can run one or more nodes on a single physical computer or[cloud](https://www.geeksforgeeks.org/cloud-computing/)server. Nodes can be either a manager or workers. The manager node dispatches units of work called tasks to worker nodes. Worker nodes receive and execute tasks dispatched from manager nodes.
* **Services:**A service is a high-level concept relating to a collection of tasks to be executed by workers. An example of a service is an [HTTP Server](https://www.geeksforgeeks.org/building-a-basic-http-server-from-scratch-in-python/) running as a [Docker Container](https://www.geeksforgeeks.org/containerization-using-docker/)on three nodes.
* **Load Balancing:** Docker includes a load balancer to process requests across all containers in the service.

## **Docker Swarm Filters**

The following are the docker swarm filters:

* **Constraints:**Based on conditions, users are restricted from creating containers on particular Docker hosts.
* **Drain Node:** If we applied Drain Nodes on any node then Docker swarm will not assign any replicas to that node.
* **Port:** Avoids the port conflicts between the application by deploying the same port applications in two different nodes.

## **Docker Swarm Mode CLI Commands**

The following are the docker swarm mode CLI commands:

* **docker swarm init:**This command is used to initialize the swarm.

docker swarm init [OPTIONS]

* **docker swarm join:**By using this command you can join a node to a swarm. The node joins as a manager node or worker node based on the token you pass with the –token flag.

docker swarm join [OPTIONS] HOST:PORT

* **docker service creates:** This is a cluster management command, and must be executed on a Swarm manager node.

docker service create [OPTIONS] IMAGE [COMMAND] [ARG...]

* **docker service inspects:**This command is used to inspect the particular service and all the details will display in JSON format.

docker service inspect [OPTIONS] SERVICE [SERVICE...]

* **docker service ls:** This command is used to see the complete list of all the services in that network.

docker service ls [OPTIONS]

* **docker service rm:** This command is used to remove the specific service you want to remove.

docker service rm SERVICE [SERVICE...]

## Docker Container vs Docker Swarm

The following are the difference between Docker Container and Docker Swarm:

| **Docker Container** | **Docker Swarm** |
| --- | --- |
| [Docker](https://www.geeksforgeeks.org/introduction-to-docker/) Container is an executable package that consists of all the code and dependencies. | Docker Swarm is the container orchestration tool that manages all the containers available in the Swarm cluster. |
| The container can be run on any[OS](https://www.geeksforgeeks.org/what-is-an-operating-system/) as Docker is an underlying runtime. | Docker Swarm will manage the cluster which consists of Docker nodes. |
| A single, isolated, and self-contained unit called a “Docker container” is capable of running an application. | You may manage many [Docker containers](https://www.geeksforgeeks.org/virtualisation-with-docker-containers/) across various servers in a cluster using the technology known as Docker Swarm. |
| Docker containers are more suitable for microservices applications than monolithic applications. | Docker Swarm will scale, deploy and balance the containers. |

## Docker Swarm vs Kubernetes

The following are the docker swarm and Kubernetes:

| **Kubernetes** | **Docker Swarm** |
| --- | --- |
| [Kubernetes](https://www.geeksforgeeks.org/introduction-to-kubernetes-k8s/) is an open-source platform used for maintaining and deploying a group of containers | Docker Swarm is designed to be a simple and lightweight container orchestration. |
| In practice, Kubernetes is most commonly used alongside Docker for better control and implantation of containerized applications. | With Docker swarm, multiple containers run on the same hardware much more efficiently than the VM environment & productivity of Docker is extremely high. |
| Applications are deployed as a combination of pods, Deployment, and services. | Apps are deployed in the form of services. |
| It supports auto-scaling of the container in a cluster. With more efficient | Docker Swarm does support auto-scaling but is not as efficient as docker swarm. |
| The health check is of two kinds: liveness and readiness. | Health checks are limited to service. |
| Hard to set up and configure. | Docker swarm setup and installation are easy. |
| It does not have extensive documentation but is quite less than Docker Swarm. But it does include everything from installation to deployment. | Docker swarm documentation is more effective, more extensive, and has even more capabilities & it includes everything from installation to deployment & quick-start instructions as well as a more detailed tutorial. |
| Kubernetes installation is provided to be quite difficult than Docker swarm and even the command for Kubernetes is quite more complex than Docker swarm. | Docker swarm installation is quite easier, by using fewer commands you can install Docker in your virtual machine or even on the cloud. |
| Azure, buffer, intel, Evernote, and Shopify Using Kubernetes. | Citizens Bank, and MetLife companies using Docker swarm. |

## Docker Service vs Docker Task

The following are the differences between docker service and docker task:

| **Aspect** | **Docker Service** | **Docker Task** |
| --- | --- | --- |
| **Definition** | A higher-level abstraction that defines how to run containers as a group | An individual instance of a container created by a service |
| **Purpose** | Manages the deployment and scaling of a group of containers | Executes a single unit of work within a service |
| **Management** | Handles orchestration, load balancing, and scaling | Represents the actual work being performed by a container |
| **Scalability** | Easily scaled up or down to manage load | Cannot be individually scaled; scaling is done at the service level |
| **State** | Desired state defined (e.g., number of replicas) | Current state of a specific container instance |
| **Failure Handling** | Automatically reschedules tasks if a container fails | Does not manage failures; relies on service for rescheduling |

## **Benefits of Docker Swarm**

The following are the benefits of Docker Swarm:

1. **Simplified Setup and Management**: Docker Swarm provides an easy-to-use and integrated toolset for orchestrating containers, making it straightforward to set up and manage a cluster of Docker nodes.
2. **Scalability**: Docker Swarm allows seamless scaling of services up or down with simple commands, enabling dynamic adjustment of resources based on demand.
3. **High Availability**: Swarm mode ensures that services are replicated across multiple nodes, providing fault tolerance and resilience by automatically redistributing tasks in case of node failures.
4. **Integrated Load Balancing**: Docker Swarm includes built-in load balancing to distribute network traffic across multiple containers, ensuring optimal performance and resource utilization.

## Docker Swarm Node – FAQs

### When should I Use Docker Swarm?

*Docker Swarm used be for simple, native Docker Orchestration purpose. It facilitates with easy setup and management for container clustering.*

### What is Swarm mode in docker?

*The Swarm mode in Docker is a native clustering and orchestrating feature that enables the deployment, management and scaling of the containers across a cluster of docker hosts.*

### Why docker swarm mode is deprecated?

*The Docker Swarm mode is deprecated because of kuberentes and it has become the more widely adopted and strong container orchestration platform leading to shift in focus and support.*

### What is docker swarm mode ingress?

*Docker Swarm mode ingres is a built-in load balancing features that routes the incoming traffic to the appropriate service across the swarm cluster.*

### Is Docker Swarm Used in Production?

*Yes Docker Swarm is used in the production environments but it is not much used as the kubernetes in the production environment. Kubernetes had more advantages then the docker swarm.*

### Is Docker Swarm Free?

*Yes Docker swarm is free to use. You can run the docker swarm on as many as docker engine instances as you want without incurring any licensing costs.*

### What Ports are Used by Docker Swarm?

* [*TCP Port*](https://www.geeksforgeeks.org/tcp-ip-ports-and-its-applications/)*for the manager node: 2377*
* *Worker node TCP port: 2376*

### Is Docker Swarm a Load Balancer?

*Docker swarm itself isn’t a full-fledged load balancer but you can achieve this by using two mechanisms.*

* *Overlay routing mesh*
* [*Ingress*](https://www.geeksforgeeks.org/what-is-kubernetes-ingress/)

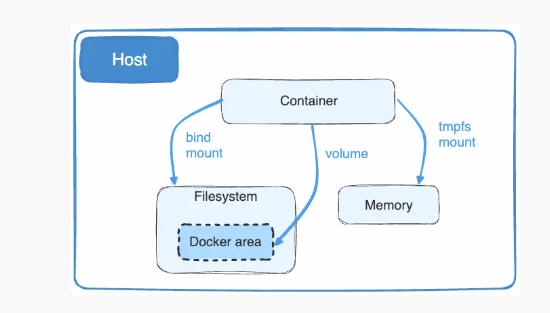
**Volumes**

Volumes are the preferred mechanism for persisting data generated by and used by Docker containers.

While [bind mounts](https://docs.docker.com/storage/bind-mounts/) are dependent on the directory structure and OS of the host machine,

volumes are completely managed by Docker. Volumes have several advantages over bind mounts:

* Volumes are easier to back up or migrate than bind mounts.
* You can manage volumes using Docker CLI commands or the Docker API.
* Volumes can be more safely shared among multiple containers.



Volumes use rprivate bind propagation, and bind propagation isn't configurable for volumes.

## [**Create and manage volumes**](https://docs.docker.com/storage/volumes/#create-and-manage-volumes)

Unlike a bind mount, you can create and manage volumes outside the scope of any container.

**Create a volume:**

$ docker volume create my-vol

**List volumes:**

$ docker volume ls

local my-vol

**Inspect a volume:**

$ docker volume inspect my-vol

[

{

"Driver": "local",

"Labels": {},

"Mountpoint": "/var/lib/docker/volumes/my-vol/\_data",

"Name": "my-vol",

"Options": {},

"Scope": "local"

}

]

**Remove a volume:**

$ docker volume rm my-vol

## [**Back up, restore, or migrate data volumes**](https://docs.docker.com/storage/volumes/#back-up-restore-or-migrate-data-volumes)

Volumes are useful for backups, restores, and migrations. Use the --volumes-from flag to create a new container that mounts that volume.

### [Back up a volume](https://docs.docker.com/storage/volumes/#back-up-a-volume)

For example, create a new container named dbstore:

$ docker run -v /dbdata --name dbstore ubuntu /bin/bash

In the next command:

* Launch a new container and mount the volume from the dbstore container
* Mount a local host directory as /backup
* Pass a command that tars the contents of the dbdata volume to a backup.tar file inside our /backup directory.

$ docker run --rm --volumes-from dbstore -v $(pwd):/backup ubuntu tar cvf /backup/backup.tar /dbdata

When the command completes and the container stops, it creates a backup of the dbdata volume.

### [Restore volume from a backup](https://docs.docker.com/storage/volumes/#restore-volume-from-a-backup)

With the backup just created, you can restore it to the same container, or to another container that you created elsewhere.

For example, create a new container named dbstore2:

$ docker run -v /dbdata --name dbstore2 ubuntu /bin/bash

Then, un-tar the backup file in the new container’s data volume:

$ docker run --rm --volumes-from dbstore2 -v $(pwd):/backup ubuntu bash -c "cd /dbdata && tar xvf /backup/backup.tar --strip 1"

You can use the techniques above to automate backup, migration, and restore testing using your preferred tools.

## [**Remove volumes**](https://docs.docker.com/storage/volumes/#remove-volumes)

A Docker data volume persists after you delete a container. There are two types of volumes to consider:

* Named volumes have a specific source from outside the container, for example, awesome:/bar.
* Anonymous volumes have no specific source. Therefore, when the container is deleted, you can instruct the Docker Engine daemon to remove them.

### [Remove anonymous volumes](https://docs.docker.com/storage/volumes/#remove-anonymous-volumes)

To automatically remove anonymous volumes, use the --rm option. For example, this command creates an anonymous /foo volume. When you remove the container, the Docker Engine removes the /foo volume but not the awesome volume.

$ docker run --rm -v /foo -v awesome:/bar busybox top

**Note**

If another container binds the volumes with --volumes-from, the volume definitions are copied and the anonymous volume also stays after the first container is removed.

### [Remove all volumes](https://docs.docker.com/storage/volumes/#remove-all-volumes)

To remove all unused volumes and free up space:

$ docker volume prune

### [Turn on static vulnerability scanning](https://docs.docker.com/docker-hub/vulnerability-scanning/#turn-on-static-vulnerability-scanning)

Repository owners and administrators can enable static vulnerability scanning on a repository. If you are a member of a Team or a Business subscription, ensure the repository you would like to enable scanning on is part of the Team or a Business tier.

To enable static vulnerability scanning:

1. Sign in to your [Docker Hub](https://hub.docker.com/) account.
2. Select **Repositories** and then choose a repository.
3. Go to the **Settings** tab.
4. Under **Image security insight settings**, select **Static scanning**.
5. Select **Save**.

### [Scan an image](https://docs.docker.com/docker-hub/vulnerability-scanning/#scan-an-image)

To scan an image for vulnerabilities, push the image to Docker Hub, to the repository for which you have turned on scanning:

1. Ensure you have installed Docker locally. See [Get Docker](https://docs.docker.com/get-docker/) to download and install Docker on your local machine.
2. Use the command line to log into your Docker account. See [docker login](https://docs.docker.com/reference/cli/docker/login/) for more information.
3. Tag the image to scan. For example, to tag a Redis image, run:
4. $ docker tag redis <your-Docker-ID>/<your-repo-name>:latest
5. Push the image to Docker Hub to trigger a static vulnerability scan for the image:

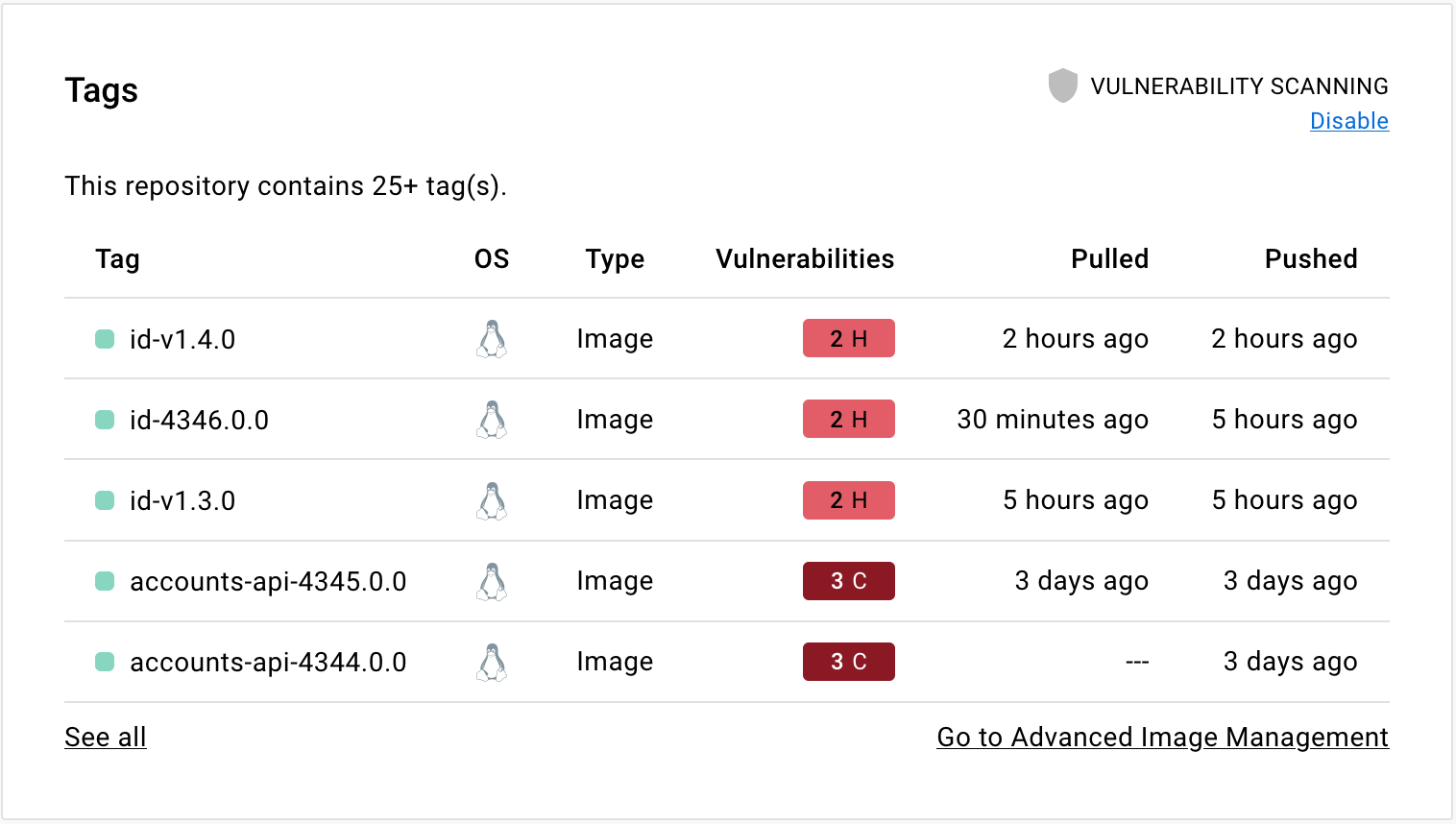
$ docker push <your-Docker-ID>/<your-repo-name>:latest

## [**View the vulnerability report**](https://docs.docker.com/docker-hub/vulnerability-scanning/#view-the-vulnerability-report)

To view the vulnerability report:

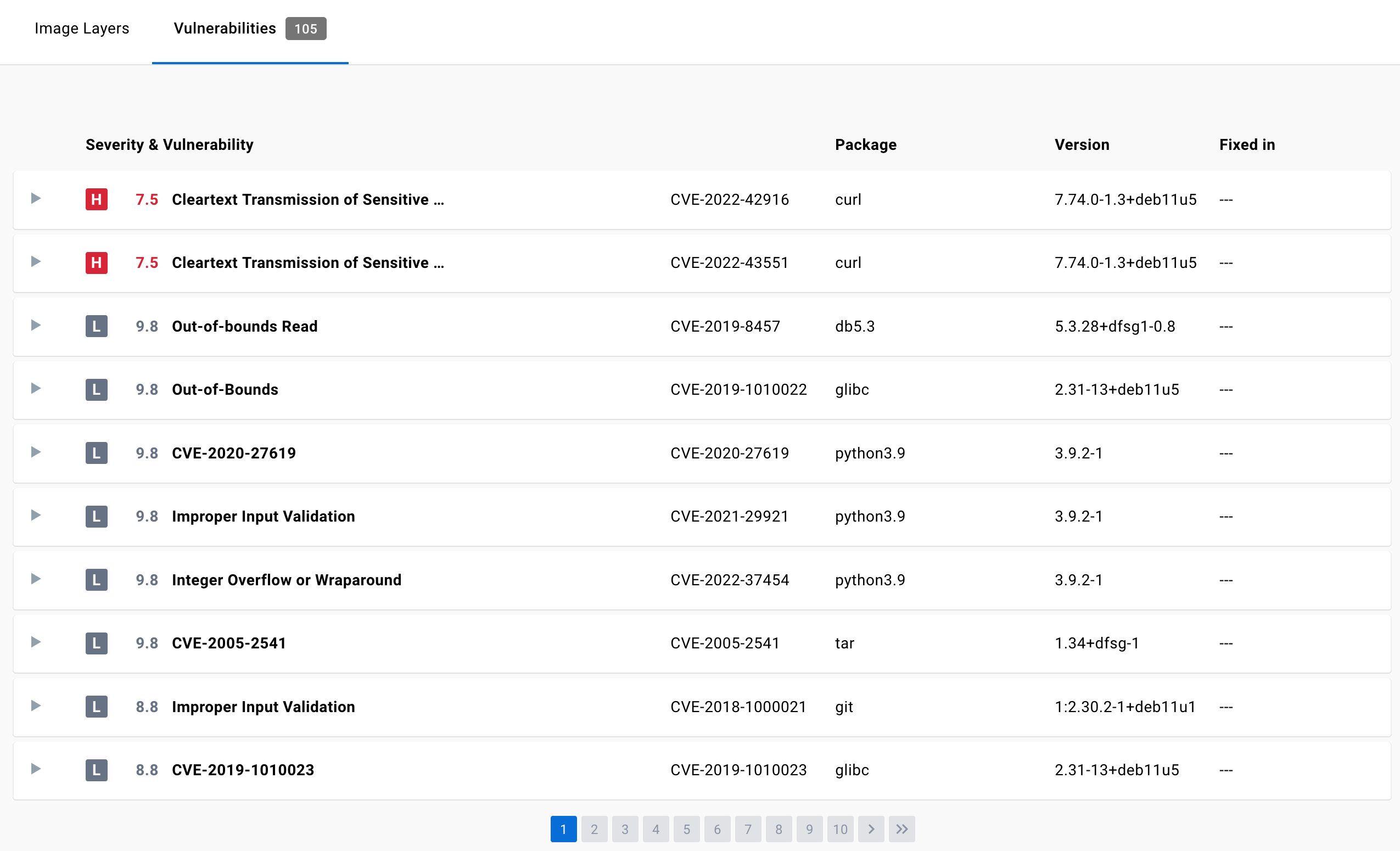
1. Go to [Docker Hub](https://hub.docker.com/) and open the repository page to view a summary of the static vulnerability scanning report.

It may take a couple of minutes for the vulnerability report to appear in your repository.



1. Select the **Tags** tab, then **Digest**, then **Vulnerabilities** to view the detailed scan report.

The scan report displays vulnerabilities identified by the scan, sorting them according to their severity, with highest severity listed at the top. It displays information about the package that contains the vulnerability, the version in which it was introduced, and whether the vulnerability is fixed in a later version.



For more information on this view, see [Image details view](https://docs.docker.com/scout/explore/image-details-view/).

### [Inspect vulnerabilities](https://docs.docker.com/docker-hub/vulnerability-scanning/#inspect-vulnerabilities)

The vulnerability report sorts vulnerabilities based on their severity. It displays information about the package that contains the vulnerability, the version in which it was introduced, and whether the vulnerability has been fixed in a later version.

The vulnerability scan report also allows development teams and security leads to compare the vulnerability counts across tags to see whether the vulnerabilities are decreasing or increasing over time.

### [Fix vulnerabilities](https://docs.docker.com/docker-hub/vulnerability-scanning/#fix-vulnerabilities)

Once a list of vulnerabilities have been identified, there are a couple of actions you can take to remediate the vulnerabilities. For example, you can:

1. Specify an updated base image in the Dockerfile, check your application-level dependencies, rebuild the Docker image, and then push the new image to Docker Hub.
2. Rebuild the Docker image, run an update command on the OS packages, and push a newer version of image to Docker Hub.
3. Edit the Dockerfile to manually remove or update specific libraries that contain vulnerabilities, rebuild the image, and push the new image to Docker Hub

Docker Scout can provide you with concrete and contextual remediation steps for improving image security. For more information, see [Docker Scout](https://docs.docker.com/scout/).

# **Security**

Docker provides security guardrails for both administrators and developers.

If you're an administrator, you can enforce sign-in across Docker products for your developers, and scale, manage, and secure your instances of Docker Desktop with DevOps security controls like Enhanced Container Isolation and Registry Access Management.

For both administrators and developers, Docker provides security-specific products such as Docker Scout, for securing your software supply chain with proactive image vulnerability monitoring and remediation strategies.

## [**For administrators**](https://docs.docker.com/security/#for-administrators)

Explore the security features Docker offers to satisfy your company's security policies.

[Settings Management](https://docs.docker.com/desktop/hardened-desktop/settings-management/)

[Learn how Settings Management can secure your developers' workflows.](https://docs.docker.com/desktop/hardened-desktop/settings-management/)

[Enhanced Container Isolation](https://docs.docker.com/desktop/hardened-desktop/enhanced-container-isolation/)

[Understand how Enhanced Container Isolation can prevent container attacks.](https://docs.docker.com/desktop/hardened-desktop/enhanced-container-isolation/)

[Registry Access Management](https://docs.docker.com/security/for-admins/registry-access-management/)

[Control the registries developers can access while using Docker Desktop.](https://docs.docker.com/security/for-admins/registry-access-management/)

[Image Access Management](https://docs.docker.com/security/for-admins/image-access-management/)

[Control the images developers can pull from Docker Hub.](https://docs.docker.com/security/for-admins/image-access-management/)

[Air-Gapped Containers](https://docs.docker.com/desktop/hardened-desktop/air-gapped-containers/)

[Restrict containers from accessing unwanted network resources.](https://docs.docker.com/desktop/hardened-desktop/air-gapped-containers/)

[Enforce sign-in](https://docs.docker.com/security/for-admins/configure-sign-in/)

[Configure sign-in for members of your teams and organizations.](https://docs.docker.com/security/for-admins/configure-sign-in/)

[Domain audit](https://docs.docker.com/security/for-admins/domain-audit/)

[Identify uncaptured users in your organization.](https://docs.docker.com/security/for-admins/domain-audit/)

[Docker Scout](https://docs.docker.com/scout/)

[Explore how Docker Scout can help you create a more secure software supply chain.](https://docs.docker.com/scout/)

[SSO](https://docs.docker.com/security/for-admins/single-sign-on/)

[Learn how to configure SSO for your company or organization.](https://docs.docker.com/security/for-admins/single-sign-on/)

[SCIM](https://docs.docker.com/security/for-admins/provisioning/scim/)

[Set up SCIM to automatically provision and deprovision users.](https://docs.docker.com/security/for-admins/provisioning/scim/)

[Roles and permissions](https://docs.docker.com/security/for-admins/roles-and-permissions/)

[Assign roles to individuals giving them different permissions within an organization.](https://docs.docker.com/security/for-admins/roles-and-permissions/)

[Private marketplace for Extensions (Early Access)](https://docs.docker.com/desktop/extensions/private-marketplace/)

[Learn how to configure and set up a private marketplace with a curated list of extensions for your Docker Desktop users.](https://docs.docker.com/desktop/extensions/private-marketplace/)

## [**For developers**](https://docs.docker.com/security/#for-developers)

See how you can protect your local environments, infrastructure, and networks without impeding productivity.

[Set up two-factor authentication](https://docs.docker.com/security/for-developers/2fa/)

[Add an extra layer of authentication to your Docker account.](https://docs.docker.com/security/for-developers/2fa/)

[Manage access tokens](https://docs.docker.com/security/for-developers/access-tokens/)

[Create personal access tokens as an alternative to your password.](https://docs.docker.com/security/for-developers/access-tokens/)

[Static vulnerability scanning](https://docs.docker.com/docker-hub/vulnerability-scanning/)

[Automatically run a point-in-time scan on your Docker images for vulnerabilities.](https://docs.docker.com/docker-hub/vulnerability-scanning/)

[Docker Engine security](https://docs.docker.com/engine/security/)

[Understand how to keep Docker Engine secure.](https://docs.docker.com/engine/security/)

[Secrets in Docker Compose](https://docs.docker.com/compose/use-secrets/)

[Learn how to use secrets in Docker Compose.](https://docs.docker.com/compose/use-secrets/)

## [**Further resources**](https://docs.docker.com/security/#further-resources)

[Security FAQs](https://docs.docker.com/faq/security/general/)

[Explore common security FAQs.](https://docs.docker.com/faq/security/general/)

[Security best practices](https://docs.docker.com/develop/security-best-practices/)

[Understand the steps you can take to improve the security of your container.](https://docs.docker.com/develop/security-best-practices/)

[Supress CVEs with VEX](https://docs.docker.com/scout/guides/vex/)

[Learn how to suppress non-applicable or fixed vulnerabilities found in your images.](https://docs.docker.com/scout/guides/vex/)