**HOW TO UPLOAD IMAGE INTO A DOCKER HUB**

Image name= demo ; tag: 0.0.1.RELEASE

Repository name= training

**Step1**: change the tag of the newly created image in order to navigate it to a particular repository so that it can uploaded to a particular repository in Docker hub of your account by using following command.

**Command**: Docker tag demo: 0.0.1.RELEASE training: v5 (user choose)

Newly created image name= training ; tag: v5

Step2: now newly created image which is navigated to the particular repository in your Docker hub account by using following command.

Command: Docker push training: v5

**HOW TO ESTABLISH CONNECTION BETWEEN TWO CONTAINERS**

**BRIDGE N/W:**

* If u create any container by default it is assigned to the bridge n/w

To establish connection between two containers that containers must be in **bridge network**.

Then by using a link we can connect two bridged network containers.

The link can be created by using below command,

**Command:** Docker run –d –p local host port num: container port num **–-env** environment variablename: link --name=container-name **–-Link** link-name repository name: tag

Tag: it represents the version of the deployable unit or repository.

**Example:** >Docker run -d -p 8100:8100 --env CURRENCY\_EXCHANGE\_SERVICE\_HOST=http://currency-exchange --name=currency-conversion --link currency-exchange in28min/currency-conversion:0.0.1-RELEASE

**NOTE:** link is not always recommended option to establish connection between two containers

Note: running version of image is called container.

**CUSTOM N/W:**

We can make communication (connection) between two containers by creating the custom network.

**Command to create custom n/w**: Docker network create network-name

Ex: Docker network create currency-network

**Container1**: currency-exchange.

**Container2**: currency-conversion.

We can assign the created n/w with the containers by using below command.

**Command1:** Docker run –d –p local host port num: container port num –name=container-name –network=network-name repository-name: tag

Ex: Docker run -d -p 8000:8000 --name=currency-exchange --network=currency-network in28min/currency-exchange:0.0.1-RELEASE

Command2: Docker run –d –p local host port num: container port num **–-env** environment variablename: link --name=container-name **–network**=network-name repository name: tag

Ex: Docker run -d -p 8100:8100 --env CURRENCY\_EXCHANGE\_SERVICE\_HOST=http: //currency-exchange --name=currency-conversion --network=currency-network in28min/currency-conversion:0.0.1-RELEASE

**Note**: the default networking mode in Docker is bridge n/w

**HOST N/W:**

The host networking driver only works on Linux hosts, but is available as a Beta feature, on Docker Desktop version 4.29 and later.

You can also use a host network for a swarm service, by passing --network host to the Docker service create command. In this case, control traffic (traffic related to managing the swarm and the service) is still sent across an overlay network, but the individual swarm service containers send data using the Docker daemon's host network and ports. This creates some extra limitations. For instance, if a service container binds to port 80, only one service container can run on a given swarm node.

## [Docker Desktop](https://docs.docker.com/network/drivers/host/#docker-desktop)

Host networking is also supported on Docker Desktop version 4.29 and later for Mac, Windows, and Linux as a beta feature. To enable this feature, navigate to the **Features in development** tab in **Settings**, and then select **Enable host networking**.

Here the commands used for communication between two containers are increasing its length. Using this type of commands format is difficult and clumsier for multiple containers.

So to overcome this difficulty we use **Docker compose**

**DOCKER COMPOSE**

Docker compose is mainly used for connecting or communicate multiple micro services each other in a simple way. Instead of using long and lengthy commands to link the containers or micro services.

# **Docker Compose overview**

Docker Compose is a tool for defining and running multi-container applications. It is the key to unlocking a streamlined and efficient development and deployment experience.

Compose simplifies the control of your entire application stack, making it easy to manage services, networks, and volumes in a single, comprehensible YAML configuration file. Then, with a single command, you create and start all the services from your configuration file.

Compose works in all environments; production, staging, development, testing, as well as CI workflows. It also has commands for managing the whole lifecycle of your application:

* Start, stop, and rebuild services
* View the status of running services
* Stream the log output of running services
* Run a one-off command on a service

Docker compose file is give below:

The give program file is a comprehensible YAML configuration file. “Docker-compose.yml”

version: '3.7'

services:

#Docker run -d -p 8000:8000 --name=currency-exchange --network=currency-network in28min/currency-exchange:0.0.1-RELEASE

  currency-exchange:

    image: in28min/currency-exchange:0.0.1-RELEASE

    ports:

      - "8000:8000"

    restart: always

    networks:

      - currency-compose-network

     # Docker run -d -p 8100:8100 --env CURRENCY\_EXCHANGE\_SERVICE\_HOST=http: //currency-exchange

     #--name=currency-conversion --network=currency-network in28min/currency-conversion:0.0.1-RELEASE

  currency-conversion:

    image: in28min/currency-conversion:0.0.1-RELEASE

    ports:

      - "8100:8100"

    restart: always

    environment:

      CURRENCY\_EXCHANGE\_SERVICE\_HOST: http://currency-exchange

    depends\_on:

      - currency-exchange

    networks:

      - currency-compose-network

# Networks to be created to facilitate communication between containers

networks:

  currency-compose-network:

Then we can launch up the two containers with mutual communication b/w them by using the below command.

Command: Docker-compose up.

In Detached mode:

Command: Docker-compose up –d.