

Container Network - Install Docker on Ubuntu



Create own Docker image

Create own Docker image

Creating and building your own Docker image involves writing a Dockerfile, which is a script that contains a series of instructions to create the image. Here's a step-by-step guide on how to do this:

Step 1: Install Docker

To install docker, we can do the following steps

Update the apt and install vim

```
sudo apt update
```

```
sudo apt install vim -y
```

Save this install.sh script file

```
#!/bin/bash
```

```
# Update package database
```

```
#!/bin/bash
```

```
# Update package database
```

```
echo "Updating package database..."
```

```
sudo apt update
```

```
# Upgrade existing packages
```

```
echo "Upgrading existing packages..."
```

```
sudo apt upgrade -y
```

```
# Install required packages
```

```
echo "Installing required packages..."
```

```
sudo apt install -y apt-transport-https ca-certificates curl software-properties-common
```

```
# Add Docker's official GPG key
```

```
echo "Adding Docker's GPG key..."
```

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o  
/usr/share/keyrings/docker-archive-keyring.gpg
```

```
# Add Docker APT repository
```

```
echo "Adding Docker APT repository..."
```

```
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]  
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee  
/etc/apt/sources.list.d/docker.list > /dev/null
```

```
# Update package database with Docker packages
```

```
echo "Updating package database with Docker packages..."
```

```
sudo apt update
```

```
# Install Docker
echo "Installing Docker..."
sudo apt install -y docker-ce

# Start Docker manually in the background
echo "Starting Docker manually in the background..."
sudo dockerd > /dev/null 2>&1 &

# Add current user to Docker group
echo "Adding current user to Docker group..."
sudo usermod -aG docker ${USER}

# Apply group changes
echo "Applying group changes..."
newgrp docker

# Set Docker socket permissions
echo "Setting Docker socket permissions..."
sudo chmod 666 /var/run/docker.sock

# Print Docker version
echo "Verifying Docker installation..."
docker --version

# Run hello-world container in the background
echo "Running hello-world container in the background..."
docker run -d hello-world

echo "Docker installation completed successfully."
echo "If you still encounter issues, please try logging out and logging back in."
```

Change the permission
 chmod +x install.sh
 Run the script file. This will install Docker on the ubuntu system.
 ./install.sh
 Open new terminal to use docker without root permission, or use in terminal 1 with root permission.

Step 2: Create a Dockerfile
 A Dockerfile is a text file that contains commands to assemble an image. Create a new directory for your project and within that directory, create a file named Dockerfile.

Step 3: Write Instructions in the Dockerfile
 Here's a basic example of a Dockerfile for a Python application:

```
# Use an official Python runtime as a parent image
FROM python:3.9-slim

# Set the working directory in the container
WORKDIR /app
```

```
# Copy the current directory contents into the container at /app
COPY . /app
```

```
# Install any needed packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt
```

```
# Make port 80 available to the world outside this container
EXPOSE 80
```

```
# Define environment variable
ENV NAME World
```

```
# Run app.py when the container launches
CMD ["python", "app.py"]
```

Step 4: Add Application Files

Add your application files in the same directory as your Dockerfile. For example, you might have:

app.py: Your main application script.

requirements.txt: A file listing the dependencies for your Python application.

Step 5: Build the Docker Image

Open a terminal and navigate to the directory containing your Dockerfile. Use the docker build command to create your image.

```
docker build -t my-python-app .
```

The -t flag tags your image with a name (my-python-app).

Step 6: Run the Docker Container

Once the image is built, you can run a container based on this image using the docker run command.

```
docker run -p 4000:80 my-python-app
```

This command maps port 80 in the container to port 4000 on your host machine.

Example: Complete Project Structure

my-python-app/

```
|— app.py
|— Dockerfile
|— requirements.txt
```

app.py

```
from flask import Flask
```

```
app = Flask(__name__)
```

```
@app.route('/')
def hello():
```

```
    return "Hello, World!"
```

```
if __name__ == '__main__':
```

```
    app.run(host='0.0.0.0', port=80)
```

requirements.txt

flask

By following these steps, you can create, build, and share your own Docker images.

Deleting docker images

Deleting a Docker image involves a few simple steps. Below is a step-by-step guide on how to remove Docker images from your system.

Step 1: List Docker Images

Before deleting an image, you need to know the image's ID or repository name. Use the following command to list all Docker images on your system:

```
docker images
```

This command will display a list of all images, including their repository names, tags, and image IDs.

Step 2: Identify the Image to Delete

From the list of images, identify the image you want to delete by noting its repository name, tag, or image ID. For example, you might see output similar to this:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
my-python-app	latest	d1e1f2b3c4d5	2 days ago	116MB
another-image	v1.0	a5b6c7d8e9f0	4 days ago	200MB

Step 3: Remove the Image

To delete an image, use the `docker rmi` command followed by the image ID or repository name and tag. For example, to remove the image `my-python-app:latest`, you can use either the image ID or the repository name and tag:

```
docker rmi d1e1f2b3c4d5
or
```

```
docker rmi my-python-app:latest
```

If the image is used by any containers, you will need to remove those containers first or use the `-f` (force) flag to forcefully remove the image:

```
docker rmi -f d1e1f2b3c4d5
```

Step 4: Verify the Image Removal

After removing the image, you can verify that it has been deleted by listing the images again:

```
docker images
```

The image you deleted should no longer appear in the list.

Additional Steps: Remove Dangling Images

Dangling images are those that are not tagged and have the repository name `<none>`. To remove all dangling images, use the following command:

```
docker image prune
```

This command will prompt you to confirm the deletion of all dangling images. To remove all unused images (not just dangling ones), add the `-a` flag:

```
docker image prune -a
```

By following these steps, you can successfully delete Docker images from your system.

explain the necessary part of the above given in a very easiest manner in details with basics also

correct if any error occur; what does these codes do with basics of each part and portion with basic example if necessary:

Awesome! Let's break everything down step-by-step in the **easiest** way possible. I'll also fix any errors and explain each part clearly like you're learning Docker for the first time.

What is Docker?

Docker is a tool that lets you **package your app** with all its files and settings so it runs **anywhere** — like a little box with everything your app needs.

Step 1: Installing Docker on Ubuntu

You're using a **Bash script** called `install.sh` to install Docker automatically.

What the script does:

1. Updates your Ubuntu packages.
2. Installs tools needed for Docker.
3. Adds Docker's security key (so Ubuntu trusts it).
4. Adds Docker's official repository (where it gets Docker from).
5. Installs Docker.
6. Starts Docker in the background.
7. Lets your user run Docker without `sudo`.
8. Runs a test container (`hello-world`).

Small Fix:

There are **two** `#!/bin/bash` lines in your script. You only need **one at the very top**.

Example (Fixed Script):

```
bash

#!/bin/bash

echo "Updating system..."
sudo apt update
sudo apt upgrade -y

echo "Installing Docker prerequisites..."
sudo apt install -y apt-transport-https ca-certificates curl software-properties-common

echo "Adding Docker GPG key..."
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings,

echo "Adding Docker repo..."
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.do

echo "Installing Docker..."
sudo apt update
```

```
sudo apt install -y docker-ce

echo "Starting Docker..."
sudo dockerd > /dev/null 2>&1 &

echo "Adding user to Docker group..."
sudo usermod -aG docker $USER
newgrp docker

echo "Setting socket permissions..."
sudo chmod 666 /var/run/docker.sock

echo "Testing Docker..."
docker --version
docker run -d hello-world

echo "Done! You may need to restart or log out/log in."
```

To Run:

```
bash

chmod +x install.sh # Make it executable
./install.sh        # Run the script
```

Step 2: Dockerfile — Your App Blueprint

This file tells Docker **how to build your app image**.

Example:

```
dockerfile

FROM python:3.9-slim          # Base image (lightweight Python)
WORKDIR /app                  # Inside container, go to /app
COPY . /app                    # Copy everything into /app
RUN pip install -r requirements.txt # Install Python packages
EXPOSE 80                      # Open port 80
ENV NAME World                 # Environment variable
CMD ["python", "app.py"]       # Run app.py when container starts
```

Step 3: Add App Files

Put these in the same folder as your Dockerfile:

`app.py`

```
python

from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello():
    return "Hello, World!"
```

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=80) # Must listen on 0.0.0.0
```

`requirements.txt`

nginx

flask

This tells Docker: install Flask to run the app.



Step 4: Build the Docker Image

Run this in the folder with the Dockerfile:

```
bash

docker build -t my-python-app .
```

- `-t` is for **tag** (naming the image).
- `.` means build using files in **current directory**.



Step 5: Run the Container

```
bash

docker run -p 4000:80 my-python-app
```

- `-p 4000:80` maps port 80 (inside container) to your computer's port 4000.
- Now go to: `http://localhost:4000` — You'll see "Hello, World!"



Step 6: Delete Docker Images

1. See all images:

```
bash

docker images
```

2. Delete by image ID or name:

```
bash

docker rmi my-python-app:latest
# or
docker rmi IMAGE_ID
```

3. Force delete if needed:

```
bash

docker rmi -f IMAGE_ID
```

4. Remove unused images:

```
bash

docker image prune      # Deletes "dangling" images
docker image prune -a   # Deletes all unused images
```

Final Project Folder Structure:

```
perl

my-python-app/
├── app.py
├── Dockerfile
└── requirements.txt
```

Summary in Simple Words:

Step	What You Do	Why
1	Install Docker	So you can use Docker tools
2	Write Dockerfile	Tells Docker how to build your app
3	Add app files	Your actual Python web app
4	Build image	Turns everything into a portable box
5	Run container	Runs your app from the box
6	Delete images	Clean up old/unused Docker images

Let me know if you want a visual diagram or a zip of this project to test directly!



only give the part which is necessary

Sure! Here's the **necessary part only**, simplified:

1. Install Docker (Ubuntu)

```
bash

sudo apt update
sudo apt install -y apt-transport-https ca-certificates curl software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings,
```



```
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) docker-ce" | sudo tee /etc/apt/sources.list.d/docker.list
sudo apt update
sudo apt install -y docker-ce
sudo usermod -aG docker $USER
newgrp docker
```

Test:

```
bash

docker --version
docker run hello-world
```

✓ 2. Create Project Folder with These Files

Example: `my-python-app`

➤ `Dockerfile`

```
dockerfile

FROM python:3.9-slim
WORKDIR /app
COPY . /app
RUN pip install -r requirements.txt
EXPOSE 80
CMD ["python", "app.py"]
```

➤ `app.py`

```
python

from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello():
    return "Hello, World!"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=80)
```

➤ `requirements.txt`

```
nginx

flask
```

✓ 3. Build Image

```
bash

docker build -t my-python-app .
```

✓ 4. Run Container

```
bash

docker run -p 4000:80 my-python-app
```

Visit: `http://localhost:4000``

✓ 5. Delete Image (If Needed)

List:

```
bash

docker images
```

Delete:

```
bash

docker rmi my-python-app
# or force:
docker rmi -f my-python-app
```

Remove unused:

```
bash

docker image prune -a
```

Let me know if you want a zip folder or Docker Compose version!



what is nc and what is Bash script?

Great questions! Let's break both down in a super simple way:

What is a Bash Script?

A **Bash script** is a file that contains a list of **commands** you normally type into the terminal — saved together so you can run them all at once.

◆ Example:

```
bash

#!/bin/bash
echo "Hello, world!"
sudo apt update
```

What's happening?

- `#!/bin/bash`: This tells the system to use **Bash** (a shell) to run the script.
- `echo`: Prints text to the terminal.
- `sudo apt update`: Updates the list of packages on your system.


Save as a file:

Save it as `script.sh`

Run it like this:

```
bash

chmod +x script.sh # Make it executable
./script.sh        # Run it
```

 Think of it like a recipe: one script = a step-by-step guide for your computer.

What is nc (Netcat)?

`nc` stands for **Netcat** — it's a simple networking tool that lets you:

- **Send or receive data** over TCP or UDP
- **Test if a port is open**
- **Create a chat between two machines**
- **Transfer files**

♦ Example: Check if a web server is running

```
bash

nc -zv google.com 80
```

- `-z`: Just scan, don't send data
- `-v`: Verbose (show details)

♦ Example: Start a listener (wait for connection)

```
bash

nc -l 1234
```

Now it listens on port 1234. If someone connects, you can chat or send data.

Summary:

Term	Meaning	Use Case
Bash Script	A file of terminal commands	Automate tasks like installing software
nc (Netcat)	Networking tool to send/receive data	Test ports, transfer files, debugging

Let me know if you want examples of using `nc` to test Docker containers or servers!