

Docker

What is Docker?

Docker is an open source platform that enables developers to build, deploy, run, update and manage containers—standardized, executable components that combine application source code with the operating system (OS) libraries and dependencies required to run that code in any environment.

- [📄 How to Mount NFS Storage for Docker Containers \(Docker Compose Version\)](#)
- [📄 How to Mount NFS Storage for Docker Containers \(with Proper Permissions\)](#)
- [📄 How to update your Docker containers safely](#)
- [📄 Mount Multiple NFS Volumes in Docker Compose](#)
- [Deploy Docker Containers on the Cloud](#)
- [How to create and manage Docker networks](#)
- [How to Use Docker Compose](#)
- [How to Mount NFS Storage for Docker Containers](#)
- [How to Mount a Local Folder into a Docker Container](#)
- [How to Backup and Restore a Docker Volume](#)
- [How to Backup and Restore Docker Volumes](#)

📄 How to Mount NFS Storage for Docker Containers (Docker Compose Version)

📄 Purpose

Configure Docker Compose to **automatically mount an NFS share** as a volume into your containers without manually mounting it on the host.

1. Prerequisites

- Install the NFS client:

bash

```
sudo apt install nfs-common -y
```

- Know your NFS server and export path.

Example:

- NFS Server: 192.168.100.11
 - Export Path: /mnt/hdd-storage/my-nfs-share
-

2. Example `docker-compose.yml`

yaml

```
version: '3.8'
```

services:

my-app:

image: my-docker-image

container_name: my-app

volumes:

- my-nfs-storage:/app/data

ports:

- "8080:8080"

restart: unless-stopped

volumes:

my-nfs-storage:

driver: local

driver_opts:

type: "nfs"

o: "addr=192.168.100.11,nfsvers=4,hard,timeo=600,retrans=2"

device: ":/mnt/hdd-storage/my-nfs-share"

3. Key Configuration Explained

“ **Note:** The colon `:` at the start of `device:` is important.

4. Deploy the Stack

Run this from the folder where `docker-compose.yml` is located:

bash

```
docker-compose up -d
```

Docker will:

- Connect to the NFS server.
- Mount the remote share.

- Bind it into the container automatically.
-

5. Best Practices

- **Let Docker manage the NFS mount** (don't mount manually with `mount` command if doing it this way).
 - Ensure your NFS server is reachable **before starting the container**.
 - Use `restart: unless-stopped` to automatically recover from network issues.
 - For production setups, use **hard mounts with retries** (`hard,timeo=600,retrans=2`).
-

6. Summary Table

📦 Bonus: Troubleshooting NF Mounts

Common issues:

- **Permission Denied**
 - Check server-side export permissions (e.g., `/etc/exports`).
 - Verify the NFS share allows the client IP.
 - Set correct UID/GID or enable `no_root_squash` if necessary.
 - **Slow Performance**
 - Try using `rsiz=131072,wsiz=131072` options.
 - Check if NFS server is overloaded.
 - Prefer NFSv4 for better performance.
 - **Mount Failures**
 - Verify NFS server IP and path.
 - Check firewall rules (port 2049 TCP/UDP for NFS).
-

📦 Final Tip

For critical workloads:

Use a private network between Docker and the NFS server for security and stability!

-
- ☐ Now this will **display perfectly** in Bookstack ☐
 - ☐ With clean titles, proper spacing, clean code blocks, and info boxes.

📁 How to Mount NFS Storage for Docker Containers (with Proper Permissions)

Purpose

Mount a remote NFS share on a Linux server for use by **Docker containers**, ensuring stable operation, correct permissions, and automatic remounting.

1. Install NFS Client on the Server

```
sudo apt update
sudo apt install nfs-common -y
```

2. Create a Local Mount Directory

Create a local directory where the NFS share will be mounted:

```
sudo mkdir -p /srv/nfs-mount
sudo chown $(whoami):$(whoami) /srv/nfs-mount
```

💡 (You can replace `/srv/nfs-mount` with your preferred path.)

3. Mount the NFS Share (Manual Test)

Example:

```
sudo mount -t nfs4 192.168.100.11:/mnt/hdd-storage/my-nfs-share /srv/nfs-mount
```

- `nfs4` : Use NFS version 4 for better performance and locking.
 - `proto=tcp` : Reliable transport protocol.
 - `hard` : Wait for server recovery instead of failing immediately.
 - `timeo=600` : Timeout setting for NFS operations.
 - `retrans=2` : Retry failed operations twice.
 - `sec=sys` : Default authentication method.
 - `_netdev` : Ensure mount occurs only after network is ready.
-

4. Verify That the Mount Worked

```
mount | grep nfs
```

You should see output like:

```
192.168.100.11:/mnt/hdd-storage/my-nfs-share on /srv/nfs-mount type nfs4 (...)
```

5. Make the Mount Persistent (Auto-Mount on Boot)

Edit your `/etc/fstab` file:

```
sudo nano /etc/fstab
```


Add the following line at the bottom:

```
192.168.100.11:/mnt/hdd-storage/my-nfs-share /srv/nfs-mount nfs4
rw,relatime,hard,proto=tcp,timeo=600,retrans=2,sec=sys,_netdev 0 0
```

Save and exit (`Ctrl+O`, `Enter`, `Ctrl+X`).

6. Test the fstab Entry Without Rebooting

```
sudo mount -a
```

No errors = success! 

7. Using the NFS Mount with Docker

When running your containers, bind-mount the NFS storage into the container:

```
docker run -d \
  --name my-container \
  -v /srv/nfs-mount:/app/data \
  my-docker-image
```

This will allow your Docker containers to directly access the NFS storage.

Summary

- **Use NFSv4** (`nfs4`) whenever possible.
- **Always** include `_netdev` in your `/etc/fstab` entries.
- **Use `hard` mounts** to protect container file operations during NFS issues.
- **Bind-mount** NFS paths carefully into containers.
- **Avoid** using NFS for database storage unless network latency is extremely low.

☐☐ Useful Commands

- Check NFS mounts: `mount | grep nfs`
 - Manual remount all: `sudo mount -a`
 - Test connectivity: `ping nfs-server-ip`
-

☐☐ Additional Notes

- If the NFS server reboots, containers **may pause** temporarily.
- If using `docker-compose.yml`, you can map volumes to `/srv/nfs-mount`.
- For production, consider creating a **systemd mount unit** for better recovery behavior.

☐☐ How to update your Docker containers safely

Purpose

Learn how to update your Docker containers safely, minimizing downtime and preventing accidental data loss.

1. Check Running Containers

List all currently running containers to know what services are active:

bash

```
docker ps
```

2. Pull the Latest Images

For each image you want to update, pull the latest version from the registry:

bash

```
docker pull imagename:tag
```

Replace `imagename:tag` with your actual image name and tag.

3. Stop and Remove Old Containers

Stop the container without deleting its data volumes:

bash

```
docker stop containername  
docker rm containername
```

Replace `containername` with your actual container name.

4. Start New Containers with Updated Images

Use your original `docker run` command or your `docker-compose` file to launch the updated containers.

bash

```
docker run -d --name containername imagename:tag
```

5. Verify Everything is Working

After starting the new containers, check that everything is healthy:

bash

```
docker ps  
docker logs containername
```

□ Summary

- **Always backup your data volumes** before updating if possible.
- **Use versioned tags** (e.g., `nginx:1.25`) instead of `latest` to avoid unexpected changes.
- **Use compose files** to automate container recreation safely.

📁 Mount Multiple NFS Volumes in Docker Compose

If you want **more than one NFS share** available to different containers, here's how:

Example: `docker-compose.yml`

yaml

```
version: '3.8'

services:
  app-one:
    image: my-first-app
    container_name: app-one
    volumes:
      - nfs-storage-one:/data
    ports:
      - "8081:8080"
    restart: unless-stopped

  app-two:
    image: my-second-app
    container_name: app-two
    volumes:
      - nfs-storage-two:/data
    ports:
      - "8082:8080"
    restart: unless-stopped

volumes:
  nfs-storage-one:
    driver: local
```

```
driver_opts:
  type: "nfs"
  o: "addr=192.168.100.11,nfsvers=4,hard,timeo=600,retrans=2"
  device: ":/mnt/hdd-storage/nfs-share-one"

nfs-storage-two:
  driver: local
  driver_opts:
    type: "nfs"
    o: "addr=192.168.100.11,nfsvers=4,hard,timeo=600,retrans=2"
    device: ":/mnt/hdd-storage/nfs-share-two"
```

☐☐ How This Works

☐☐ Quick Commands

To start everything:

```
bash
```

[Copy](#) [Edit](#)

```
docker-compose up -d
```

To check if volumes are correctly mounted:

```
bash
```

[Copy](#) [Edit](#)

```
docker volume ls
```

```
docker inspect <volume_name>
```

☐ **This allows you to have multiple apps, each with their own NFS storage**, clean and scalable.

☐ **No manual mount commands needed** on the host!

Deploy Docker Containers on the Cloud

image.png
image not found or type unknown

Objective:

In this Article we want you to be able to deploy secure apps with containers in the cloud space. This deployment is useful for deploying apps or services without the need of a separate Firewall on prem or the cloud to secure and encrypt the communication. Another benefits of this design is, that the user's never interact with the actual server.

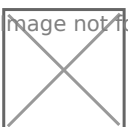
Pre-Requisites:

- Domain or be able to create sub-domains (cloud-npm.your_domain.tld, cloud-portainer.your_domain.tld, any_app_container.your_domain.tld)
- Ability to open ports 80,81,443 on the firewall or VPS
- Ubuntu Linux VM (with the necessary resources to run your containers)

This guide will **not** show you how to create a VM, open the necessary ports on your VPS (80, 81, 443), or create the necessary DNS records on your Public DNS Server or Domain Registrar.

Understanding the Design:

image.png
image not found or type unknown



The above shows how we intend the communication to work. In this design, you will always use Nginx Proxy manager (NPM), and to manage docker from a web-ui we are going to use Portainer. We will expose to the public port **80 TCP** and port **443 TCP**. We will also open temporarily port **81 TCP** for NPM's initial setup / management.

Install Docker using the convenience script

Docker provides a convenience script at https://get.docker.com/open_in_new to install Docker into development environments non-interactively. The convenience script isn't recommended for production environments, but it's useful for creating a provisioning script tailored to your needs.

Also refer to the [install using the repository](#) steps to learn about installation steps to install using the package repository. The source code for the script is open source, and you can find it in the [docker-install repository on GitHub](#)[open_in_new](#).

Always examine scripts downloaded from the internet before running them locally. Before installing, make yourself familiar with potential risks and limitations of the convenience script:

- The script requires `root` or `sudo` privileges to run.
- The script attempts to detect your Linux distribution and version and configure your package management system for you.
- The script doesn't allow you to customize most installation parameters.
- The script installs dependencies and recommendations without asking for confirmation. This may install a large number of packages, depending on the current configuration of your host machine.
- By default, the script installs the latest stable release of Docker, containerd, and runc. When using this script to provision a machine, this may result in unexpected major version upgrades of Docker. Always test upgrades in a test environment before deploying to your production systems.
- The script isn't designed to upgrade an existing Docker installation. When using the script to update an existing installation, dependencies may not be updated to the expected version, resulting in outdated versions.

Tip: preview script steps before running

You can run the script with the `--dry-run` option to learn what steps the script will run when invoked:

[copy](#)

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh ./get-docker.sh --dry-run
```

This example downloads the script from https://get.docker.com/open_in_new and runs it to install the latest stable release of Docker on Linux:

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
Executing docker install script, commit: 7cae5f8b0decc17d6571f9f52eb840fbc13b2737
<...>
```

Linux post-installation steps for Docker Engine

These optional post-installation procedures describe how to configure your Linux host machine to work better with Docker.

Manage Docker as a non-root user

The Docker daemon binds to a Unix socket, not a TCP port. By default it's the `root` user that owns the Unix socket, and other users can only access it using `sudo`. The Docker daemon always runs as the `root` user.

If you don't want to preface the `docker` command with `sudo`, create a Unix group called `docker` and add users to it. When the Docker daemon starts, it creates a Unix socket accessible by members of the `docker` group. On some Linux distributions, the system automatically creates this group when installing Docker Engine using a package manager. In that case, there is no need for you to manually create the group.

Warning

The `docker` group grants root-level privileges to the user. For details on how this impacts security in your system, see [Docker Daemon Attack Surface](#).

Note To run Docker without root privileges, see [Run the Docker daemon as a non-root user \(Rootless mode\)](#).

To create the `docker` group and add your user:

1. Create the `docker` group.

```
sudo groupadd docker
```

2. Add your user to the `docker` group.

```
sudo usermod -aG docker $USER
```

3. Log out and log back in so that your group membership is re-evaluated.

If you're running Linux in a virtual machine, it may be necessary to restart the virtual machine for changes to take effect.

You can also run the following command to activate the changes to groups:

```
newgrp docker
```

4. Verify that you can run `docker` commands without `sudo`.

```
docker run hello-world
```

This command downloads a test image and runs it in a container. When the container runs, it prints a message and exits.

If you initially ran Docker CLI commands using `sudo` before adding your user to the `docker` group, you may see the following error:

```
WARNING: Error loading config file: /home/user/.docker/config.json -  
stat /home/user/.docker/config.json: permission denied
```

This error indicates that the permission settings for the `~/.docker/` directory are incorrect, due to having used the `sudo` command earlier.

To fix this problem, either remove the `~/.docker/` directory (it's recreated automatically, but any custom settings are lost), or change its ownership and permissions using the following commands:

```
sudo chown "$USER":"$USER" /home/"$USER"/.docker -R  
sudo chmod g+rx "$HOME/.docker" -R
```

Configure Docker to start on boot with systemd

Many modern Linux distributions use [systemd](#) to manage which services start when the system boots. On Debian and Ubuntu, the Docker service starts on boot by default. To automatically start Docker and containerd on boot for other Linux distributions using systemd, run the following commands:

```
sudo systemctl enable docker.service
sudo systemctl enable containerd.service
```

To stop this behavior, use `disable` instead.

```
sudo systemctl disable docker.service
sudo systemctl disable containerd.service
```

If you need to add an HTTP proxy, set a different directory or partition for the Docker runtime files, or make other customizations, see [customize your systemd Docker daemon options](#).

Install Docker-Compose standalone

Important

From July 2023 Compose V1 stopped receiving updates. It's also no longer available in new releases of Docker Desktop.

Compose V2 is included with all currently supported versions of Docker Desktop. For more information, see [Migrate to Compose V2](#). Docker's documentation refers to and describes Compose V2 functionality.

On this page you can find instructions on how to install Compose standalone on Linux or Windows Server, from the command line.

On Linux

Compose standalone

Note that Compose standalone uses the `-compose` syntax instead of the current standard syntax `compose`.

For example type `docker-compose up` when using Compose standalone, instead of `docker compose up`.

1. To download and install Compose standalone, run:

```
curl -SL https://github.com/docker/compose/releases/download/v2.23.0/docker-compose-linux-x86_64 -o /usr/local/bin/docker-compose
```

2. Apply executable permissions to the standalone binary in the target path for the installation.
3. Test and execute compose commands using `docker-compose`.

“**Tip**

If the command `docker-compose` fails after installation, check your path. You can also create a symbolic link to `/usr/bin` or any other directory in your path. For example:

```
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

Creating a new Docker Network

I'm going to demonstrate how to create a bridge network and then show you how to deploy a container on that network. We'll create a network called net-to-proxy. The creation of this network can be achieved with a single command:

```
docker network create --driver bridge net-to-proxy
```

The output of that command will be a long string of characters that represents the ID of that newly-created network

Organization:

Because we will be deploying Three **Stacks** with docker-compose, we will need to keep our files organized. We are going to use the "mkdir" command to create our directories for the containers in the "home" folder of the current user.

```
mkdir ./npm && mkdir ./portainer && mkdir ./bookstack
```

The plan is to have a "docker-compose.yml" file for each stack in their own folders.

Prepare Deployment of Portainer

Portainer has a few dependencies that must be supplied when you start your container:

- It requires a volume to store persistent data.
- Your host's Docker socket should be mounted into the container so that Portainer can access and interact with the [Docker daemon](#).
- You need to bind a port to the container so you can access the web UI.

Change to the Portainer directory we created `~/portainer`. Create a file called "docker-compose.yml". This lets you write the container's configuration into a file so you can bring up the app with a single command.

Portainer-Docker-Compose.yml

```
version: "3"
services:
  portainer:
    image: portainer/portainer-ce:latest
    ports:
      - 9443:9443
    volumes:
      - data:/data
      - /var/run/docker.sock:/var/run/docker.sock
    restart: unless-stopped
    networks:
      - portainer_net
      - net-to-proxy
volumes:
  data: ./poertainer_data

networks:
  portainer_net:
    driver: bridge # This means that, this network is using the host bridge
  net-to-proxy:
```

```
external: true # This means that, this network was created previously and is external
```

Here, the `image` field is set to `portainer/portainer-ce:latest` to use the latest Portainer CE release from Docker Hub. Change this to `portainer/portainer-ee:latest` if you've purchased an Enterprise Edition license.

The `ports` field sets up a port binding from your host to the container. You'll be able to access the Portainer UI by visiting `https://localhost:9443`. Portainer provides a self-signed HTTPS certificate, which you can override by [mounting your own](#) into the container.

The `volumes` field sets up a `data` volume that's mounted to `/data` inside the container. Portainer will write your settings to this location, allowing them to persist after the container restarts. The host's Docker socket, `/var/run/docker.sock`, is [bind mounted](#) straight into the container so Portainer can manage the Docker installation it's running within.

The `Networks` field, maps out the networks the container will have. There are other ways to add a network to a stack, in this case we used this way to ensure connectivity after reboots.

Finally, the `restart` field is set to `unless-stopped`, so Docker automatically starts Portainer after the host reboots unless you manually stop the container first.

Now you can use this Compose file to bring up Portainer:

Please **DO NOT RUN THE COMMAND BELOW**- yet

```
docker compose up -d
```

NGINX Proxy Manager Deployment

Compatilibility

The docker images support the following architectures:

- amd64
- arm64
- armv7

The docker images are a manifest of all the architecture docker builds supported, so this means you don't have to worry about doing anything special and you can follow the common instructions above.

Check out the [dockerhub tags](#) for a list of supported architectures and if you want one that doesn't exist, [create a feature request](#).

Also, if you don't know how to already, follow [this guide to install docker and docker-compose](#) on Raspbian.

Please note that the `jc21/mariadb-aria:latest` image might have some problems on some ARM devices, if you want a separate database container, use the `yobasystems/alpine-mariadb:latest` image.

Navigate to `~/npm` and create a `docker-compose.yml` file:

NGINX-Docker-Compose.yml

```
version: '3.8'

services:
  app:
    image: 'jc21/nginx-proxy-manager:latest'
    restart: unless-stopped
    ports:
      # These ports are in format <host-port>:<container-port>
      - '80:80' # Public HTTP Port
      - '443:443' # Public HTTPS Port
      - '81:81' # Admin Web Port
      # Add any other Stream port you want to expose
      # - '21:21' # FTP
    environment:
      # Mysql/Maria connection parameters:
      DB_MYSQL_HOST: "db"
      DB_MYSQL_PORT: 3306
      DB_MYSQL_USER: "npm"
      DB_MYSQL_PASSWORD: "npm" #change this line
      DB_MYSQL_NAME: "npm"
      # Uncomment this if IPv6 is not enabled on your host
      # DISABLE_IPV6: 'true'
```

```
volumes:
  - ./data:/data
  - ./letsencrypt:/etc/letsencrypt
depends_on:
  - db
networks:
  - net-to-proxy
  - npm_default
db:
  image: 'jc21/mariadb-aria:latest'
  restart: unless-stopped
  environment:
    MYSQL_ROOT_PASSWORD: 'npm' #change this line
    MYSQL_DATABASE: 'npm'
    MYSQL_USER: 'npm'
    MYSQL_PASSWORD: 'npm' # change this line
  volumes:
    - ./mysql:/var/lib/mysql
  networks:
    - npm_default
networks:
  net-to-proxy:
    external: true
  npm_default:
    driver: bridge
```

Please note the lines that need to be edited

Then:

```
docker-compose up -d
```

Initial Run

After the app is running for the first time, the following will happen:

1. GPG keys will be generated and saved in the data folder

2. The database will initialize with table structures
3. A default admin user will be created

This process can take a couple of minutes depending on your machine.

Next Navigate on a web-browser to `https://your_domain.tld:81`

This is where exposing the port **81 TCP** is necessary and Temporary.

image.png
Image could not be loaded or type unknown

Default Administrator User

Email: admin@example.com

Password: changeme

Immediately after logging in with this default user you will be asked to modify your details and change your password.

Proxy NGINX Management Dashboard

Deploy Portainer

Now, that NGINX Proxy Manager is running, we can navigate to the portainer folder `~/portainer` and start portainer with:

```
docker-compose up -d
```

next, run:

```
docker ps -a
```

This command is useful to see all running docker containers, the container's ID and container's open ports to the Docker host.

find portainer's IP address by running:

```
docker inspect --format='{{.NetworkSettings.IPAddress}}' <CONTAINER ID>
```

or

```
docker inspect <CONTAINER ID>
```

Here is an Example output:

Code example

```
ubuntu@instance-20231025-1926:~$ docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
91ba49cd0990   lscr.io/linuxserver/bookstack      "/init"                 27 hours ago  Up 27 hours   443/tcp,
0.0.0.0:6875->80/tcp, :::6875->80/tcp
60a49a71795a   lscr.io/linuxserver/mariadb        "/init"                 27 hours ago  Up 27 hours   3306/tcp
f756a302bcbd   jc21/nginx-proxy-manager:latest    "/init"                 27 hours ago  Up 27 hours   0.0.0.0:80-81-
>80-81/tcp, :::80-81->80-81/tcp, 0.0.0.0:443->443/tcp, :::443->443/tcp
1ed7856bd307   jc21/mariadb-aria:latest           "/scripts/run.sh"       27 hours ago  Up 27 hours   3306/tcp
ddccc9f35392   portainer/portainer-ce:latest      "/portainer"            2 days ago    Up 29 hours   0.0.0.0:8000-
>8000/tcp, :::8000->8000/tcp, 0.0.0.0:9443->9443/tcp, :::9443->9443/tcp, 9000/tcp
ubuntu@instance-20231025-1926:~$ docker inspect ddccc9f35392
[
  {
    "Id": "ddccc9f35392eed4e829f7cf0b3d17e79a9af5a4eb57232306ab10355cfd55c7",
    "Created": "2023-10-26T03:21:27.100878836Z",
    "Path": "/portainer",
    "Args": [],
    "State": {
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
```

```
    "Pid": 7701,
    "ExitCode": 0,
    "Error": "",
    "StartedAt": "2023-10-27T15:40:38.580190785Z",
    "FinishedAt": "2023-10-27T15:40:37.763361341Z"
  },
  "Image": "sha256:ecc519e8696aa56565c7b34ea94ae31d25325ebdb4d91077c30d6b9f757ae7cd",
  "ResolvConfPath":
"/var/lib/docker/containers/ddccc9f35392eed4e829f7cf0b3d17e79a9af5a4eb57232306ab10355cfd55c7/resolv.conf",
  "HostnamePath":
"/var/lib/docker/containers/ddccc9f35392eed4e829f7cf0b3d17e79a9af5a4eb57232306ab10355cfd55c7/hostname",
  "HostsPath":
"/var/lib/docker/containers/ddccc9f35392eed4e829f7cf0b3d17e79a9af5a4eb57232306ab10355cfd55c7/hosts",
  "LogPath":
"/var/lib/docker/containers/ddccc9f35392eed4e829f7cf0b3d17e79a9af5a4eb57232306ab10355cfd55c7/ddccc9f35392eed4e829f7cf0b3d17e79a9af5a4eb57232306ab10355cfd55c7-json.log",
  "Name": "/portainer",
  "RestartCount": 0,
  "Driver": "overlay2",
  "Platform": "linux",
  "MountLabel": "",
  "ProcessLabel": "",
  "AppArmorProfile": "docker-default",
  "ExecIDs": null,
  "HostConfig": {
    "Binds": [
      "/var/run/docker.sock:/var/run/docker.sock",
      "portainer_data:/data"
    ],
    "ContainerIDFile": "",
    "LogConfig": {
      "Type": "json-file",
      "Config": {}
    },
    "NetworkMode": "default",
    "PortBindings": {
```

```
"8000/tcp": [  
  {  
    "HostIp": "",  
    "HostPort": "8000"  
  }  
],  
"9443/tcp": [  
  {  
    "HostIp": "",  
    "HostPort": "9443"  
  }  
]  
},  
"RestartPolicy": {  
  "Name": "always",  
  "MaximumRetryCount": 0  
},  
"AutoRemove": false,  
"VolumeDriver": "",  
"VolumesFrom": null,  
"ConsoleSize": [  
  83,  
  205  
],  
"CapAdd": null,  
"CapDrop": null,  
"CgroupnsMode": "private",  
"Dns": [],  
"DnsOptions": [],  
"DnsSearch": [],  
"ExtraHosts": null,  
"GroupAdd": null,  
"IpcMode": "private",  
"Cgroup": "",  
"Links": null,  
"OomScoreAdj": 0,  
"PidMode": "",  
"Privileged": false,  
"PublishAllPorts": false,
```

```
"ReadonlyRootfs": false,
"SecurityOpt": null,
"UTSMode": "",
"UsersnsMode": "",
"ShmSize": 67108864,
"Runtime": "runc",
"Isolation": "",
"CpuShares": 0,
"Memory": 0,
"NanoCpus": 0,
"CgroupParent": "",
"BlkioWeight": 0,
"BlkioWeightDevice": [],
"BlkioDeviceReadBps": [],
"BlkioDeviceWriteBps": [],
"BlkioDeviceReadIOps": [],
"BlkioDeviceWriteIOps": [],
"CpuPeriod": 0,
"CpuQuota": 0,
"CpuRealtimePeriod": 0,
"CpuRealtimeRuntime": 0,
"CpusetCpus": "",
"CpusetMems": "",
"Devices": [],
"DeviceCgroupRules": null,
"DeviceRequests": null,
"MemoryReservation": 0,
"MemorySwap": 0,
"MemorySwappiness": null,
"OomKillDisable": null,
"PidsLimit": null,
"Ulimits": null,
"CpuCount": 0,
"CpuPercent": 0,
"IOMaximumIOps": 0,
"IOMaximumBandwidth": 0,
"MaskedPaths": [
  "/proc/asound",
  "/proc/acpi",
```

```
    "/proc/kcore",
    "/proc/keys",
    "/proc/latency_stats",
    "/proc/timer_list",
    "/proc/timer_stats",
    "/proc/sched_debug",
    "/proc/scsi",
    "/sys/firmware"
```

```
],
```

```
"ReadonlyPaths": [
```

```
    "/proc/bus",
    "/proc/fs",
    "/proc/irq",
    "/proc/sys",
    "/proc/sysrq-trigger"
```

```
]
```

```
},
```

```
"GraphDriver": {
```

```
    "Data": {
```

```
        "LowerDir":
```

```
"/var/lib/docker/overlay2/6c52adb809a69f5cd0943b6e0783b0765f0a2e16cb8e44f5929a04fa38610d03-
init/diff:/var/lib/docker/overlay2/c91f7fecf896c5c66a30b63d79aaf79127d7d1676ce0c56e9b88b63f4e14bbb3
/diff:/var/lib/docker/overlay2/6ca723844f37bba15380d5f7ee1f92b6370960933e95e8f12c18cb0695263b3e/d
iff:/var/lib/docker/overlay2/dbfe93b7499b1686e55cd83ee1c6a3d994452e3e8e082fa6c812c68f048ace8e/diff
:/var/lib/docker/overlay2/c63d3db485f14dc788088a1f548f973ec867f68f39c36ff587b069f89c346429/diff:/va
r/lib/docker/overlay2/fd595bcd1e7587be951eb9f4125b451669a139441fc927f87ededd3f2fda02a7/diff:/var/li
b/docker/overlay2/de72a959af4301fbfd1bb57a3b504654551de204ccd208c0849b8e6f632ec6db/diff:/var/lib/
docker/overlay2/656ed3a41d736d4e22245ff7c5a7e702d8c56a8c248c61abd8d7cb9a0801042c/diff:/var/lib/d
ocker/overlay2/a44742a1ea16001b049ea1d135856a4a5c757a165c4decac6cb8078aac42e2ac/diff:/var/lib/d
ocker/overlay2/a0e0f2c89d00e6fc95f73869f237933eef4b9f126119cec0c7b8f14b1a5fd57d/diff:/var/lib/docke
r/overlay2/08b663333a2880aff2a507afd630f680362f60eb7d581045845dac6354440943/diff:/var/lib/docker/
overlay2/107036274e659999f78fef4379be3bcc4e50107c5fbd030e27757a1b4d78a836/diff",
```

```
        "MergedDir":
```

```
"/var/lib/docker/overlay2/6c52adb809a69f5cd0943b6e0783b0765f0a2e16cb8e44f5929a04fa38610d03/mer
ged",
```

```
        "UpperDir":
```

```
"/var/lib/docker/overlay2/6c52adb809a69f5cd0943b6e0783b0765f0a2e16cb8e44f5929a04fa38610d03/diff",
```

```
        "WorkDir":
```

```
"/var/lib/docker/overlay2/6c52adb809a69f5cd0943b6e0783b0765f0a2e16cb8e44f5929a04fa38610d03/work
```

"

```
    },
    "Name": "overlay2"
  },
  "Mounts": [
    {
      "Type": "volume",
      "Name": "portainer_data",
      "Source": "/var/lib/docker/volumes/portainer_data/_data",
      "Destination": "/data",
      "Driver": "local",
      "Mode": "z",
      "RW": true,
      "Propagation": ""
    },
    {
      "Type": "bind",
      "Source": "/var/run/docker.sock",
      "Destination": "/var/run/docker.sock",
      "Mode": "",
      "RW": true,
      "Propagation": "rprivate"
    }
  ],
  "Config": {
    "Hostname": "ddccc9f35392",
    "Domainname": "",
    "User": "",
    "AttachStdin": false,
    "AttachStdout": false,
    "AttachStderr": false,
    "ExposedPorts": {
      "8000/tcp": {},
      "9000/tcp": {},
      "9443/tcp": {}
    },
    "Tty": false,
    "OpenStdin": false,
    "StdinOnce": false,
```

```

"Env": [
  "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
],
"Cmd": null,
"Image": "portainer/portainer-ce:latest",
"Volumes": {
  "/data": {}
},
"WorkingDir": "/",
"Entrypoint": [
  "/portainer"
],
"OnBuild": null,
"Labels": {
  "com.docker.desktop.extension.api.version": ">= 0.2.2",
  "com.docker.desktop.extension.icon": "https://portainer-io-
assets.sfo2.cdn.digitaloceanspaces.com/logos/portainer.png",
  "com.docker.extension.additional-urls":
"[{\\"title\\":\\"Website\\",\\"url\\":\\"https://www.portainer.io?utm_campaign=DockerCon&utm_source=DockerDes
ktop\\"},{\\"title\\":\\"Documentation\\",\\"url\\":\\"https://docs.portainer.io\\"},{\\"title\\":\\"Support\\",\\"url\\":\\"https://j
oin.slack.com/t/portainer/shared_invite/zt-txh3ljab-52QHTyjCqbe5RibC2lcjKA\\"}]",
  "com.docker.extension.detailed-description": "<p data-renderer-start-
pos=\\\"226\\\">Portainer's Docker Desktop extension gives you access to all of Portainer's rich
management functionality within your docker desktop experience.</p><h2 data-renderer-start-
pos=\\\"374\\\">With Portainer you can:</h2><ul><li>See all your running containers</li><li>Easily view all
of your container logs</li><li>Console into containers</li><li>Easily deploy your code into containers
using a simple form</li><li>Turn your YAML into custom templates for easy reuse</li></ul><h2 data-
renderer-start-pos=\\\"660\\\">About Portainer&nbsp;</h2><p data-renderer-start-pos=\\\"680\\\">Portainer is
the world's most popular universal container management platform with more than 650,000 active
monthly users. Portainer can be used to manage Docker Standalone, Kubernetes, Docker Swarm and Nomad
environments through a single common interface. It includes a simple GitOps automation engine and a Kube
API.&nbsp;</p><p data-renderer-start-pos=\\\"1006\\\">Portainer Business Edition is our fully supported
commercial grade product for business-wide use. It includes all the functionality that businesses need to
manage containers at scale. Visit <a class=\\\"sc-jKJIte dPfAtb\\\" href=\\\"http://portainer.io/\\\"
title=\\\"http://Portainer.io\\\" data-renderer-mark=\\\"true\\\">Portainer.io</a> to learn more about Portainer
Business and <a class=\\\"sc-jKJIte dPfAtb\\\" href=\\\"http://portainer.io/take-
3?utm_campaign=DockerCon&utm_source=Docker%20Desktop\\\" title=\\\"http://portainer.io/take-
3?utm_campaign=DockerCon&utm_source=Docker%20Desktop\\\" data-renderer-mark=\\\"true\\\">get 3
free nodes.</a></p>",

```



```

        "com.docker.extension.publisher-url": "https://www.portainer.io",
        "com.docker.extension.screenshots": "[{\\"alt\\": \\"screenshot one\\", \\"url\\": \\"https://portainer-io-assets.sfo2.digitaloceanspaces.com/screenshots/docker-extension-1.png\\"}, {\\"alt\\": \\"screenshot two\\", \\"url\\": \\"https://portainer-io-assets.sfo2.digitaloceanspaces.com/screenshots/docker-extension-2.png\\"}, {\\"alt\\": \\"screenshot three\\", \\"url\\": \\"https://portainer-io-assets.sfo2.digitaloceanspaces.com/screenshots/docker-extension-3.png\\"}, {\\"alt\\": \\"screenshot four\\", \\"url\\": \\"https://portainer-io-assets.sfo2.digitaloceanspaces.com/screenshots/docker-extension-4.png\\"}, {\\"alt\\": \\"screenshot five\\", \\"url\\": \\"https://portainer-io-assets.sfo2.digitaloceanspaces.com/screenshots/docker-extension-5.png\\"}, {\\"alt\\": \\"screenshot six\\", \\"url\\": \\"https://portainer-io-assets.sfo2.digitaloceanspaces.com/screenshots/docker-extension-6.png\\"}, {\\"alt\\": \\"screenshot seven\\", \\"url\\": \\"https://portainer-io-assets.sfo2.digitaloceanspaces.com/screenshots/docker-extension-7.png\\"}, {\\"alt\\": \\"screenshot eight\\", \\"url\\": \\"https://portainer-io-assets.sfo2.digitaloceanspaces.com/screenshots/docker-extension-8.png\\"}, {\\"alt\\": \\"screenshot nine\\", \\"url\\": \\"https://portainer-io-assets.sfo2.digitaloceanspaces.com/screenshots/docker-extension-9.png\\"}]",
        "io.portainer.server": "true",
        "org.opencontainers.image.description": "Docker container management made simple, with the world's most popular GUI-based container management platform.",
        "org.opencontainers.image.title": "Portainer",
        "org.opencontainers.image.vendor": "Portainer.io"
    }
},
"NetworkSettings": {
    "Bridge": "",
    "SandboxID": "043d93a4ae7e151ea78b088376579e29e7b0efe33dd961cffba0307c642a04ca",
    "HairpinMode": false,
    "LinkLocalIPv6Address": "",
    "LinkLocalIPv6PrefixLen": 0,
    "Ports": {
        "8000/tcp": [
            {
                "HostIp": "0.0.0.0",
                "HostPort": "8000"
            },
            {
                "HostIp": ":::",
                "HostPort": "8000"
            }
        ]
    }
},
],

```

```
"9000/tcp": null,
"9443/tcp": [
  {
    "HostIp": "0.0.0.0",
    "HostPort": "9443"
  },
  {
    "HostIp": "::",
    "HostPort": "9443"
  }
],
"SandboxKey": "/var/run/docker/netns/043d93a4ae7e",
"SecondaryIPAddresses": null,
"SecondaryIPv6Addresses": null,
"EndpointID": "6357a139fd6b5f3429a595294a79ee409b1d45c360e0c3a71753c6e2986b2370",
"Gateway": "172.17.0.1",
"GlobalIPv6Address": "",
"GlobalIPv6PrefixLen": 0,
"IPAddress": "172.17.0.2",
"IPPrefixLen": 16,
"IPv6Gateway": "",
"MacAddress": "02:42:ac:11:00:02",
"Networks": {
  "bridge": {
    "IPAMConfig": null,
    "Links": null,
    "Aliases": null,
    "NetworkID": "bbea4213fff37a9904b60d7e2699721e1311e11f770e2095288875175cfdb533",
    "EndpointID": "6357a139fd6b5f3429a595294a79ee409b1d45c360e0c3a71753c6e2986b2370",
    "Gateway": "172.17.0.1",
    "IPAddress": "172.17.0.2",
    "IPPrefixLen": 16,
    "IPv6Gateway": "",
    "GlobalIPv6Address": "",
    "GlobalIPv6PrefixLen": 0,
    "MacAddress": "02:42:ac:11:00:02",
    "DriverOpts": null
  },

```

```
"towne-net": {
  "IPAMConfig": null,
  "Links": null,
  "Aliases": [
    "ddccc9f35392"
  ],
  "NetworkID": "24fba8d25afa8fbe7efec42df03de1844cc72197ed22bfa137c6120cffc2c3db",
  "EndpointID": "e7501abfa77e30abc66313c1e2c25eef4cf2f5c6d73baf5ed5107d03d312a6c8",
  "Gateway": "172.18.0.1",
  "IPAddress": "172.18.0.4",
  "IPPrefixLen": 16,
  "IPv6Gateway": "",
  "GlobalIPv6Address": "",
  "GlobalIPv6PrefixLen": 0,
  "MacAddress": "02:42:ac:12:00:04",
  "DriverOpts": null
}
}
}
}
]
ubuntu@instance-20231025-1926:~$
```

The above is an example. This command gives a lot of information. Look Carefully for the `"IPv4Address";` line.

[image.png](#) or type unknown

Proxy NPM's Management Interface

With your service running, return to the Nginx Proxy Manager interface. There, add a proxy host for the service, creating a reverse proxy that forwards traffic from the domain to the service.

npm-1.png
Image not found or type unknown

image.png
Image not found or type unknown

- Navigate to the **Proxy Hosts** page. Get there either using the **Proxy Hosts** button from the **Dashboard** or via the **Hosts > Proxy Hosts** option from the top menu bar.
- Click the **Add Proxy Host** button. Complete the form that displays as follows:
 - Enter the domain name to be used for your service in the **Domain Names** field.
 - Set the **Scheme** to *https*. This refers to the scheme used by Nginx to access the service, not the scheme used for the proxy itself. A later step adds SSL encryption to the proxy.
 - Enter the service address in the **Forward Hostname/IP** field. For this case **127.0.0.1**.
Enter the local-docker address from which the proxy manager could access the service.
 - Enter the service port in the **Forward Port** field. Following the configuration for NGiNx used in this tutorial, that port is 81.
 - Toggle on the **Block Common Exploits, Cache Assets, Websockets Support** options as this is generally a nice feature to have.
 - Leave the remaining fields at their defaults.

image.png
Image not found or type unknown

- Before saving the configuration, navigate to the **SSL** tab and complete the form as follows:
 - Select *Request a new SSL Certificate* from the **SSL Certificate** drop down.
 - Toggle on the **Force SSL** option to ensure HTTPS is used, encrypting traffic to and from the service.
 - Enter an email address for the Let's Encrypt certificate process. Let's Encrypt uses this to alert you when the certificate needs to be renewed.
 - Select the **I Agree** toggle after reading the terms of service for Let's Encrypt.
 - Leave the remaining fields at their defaults.

Adding an SSL certificate to a proxy host in the Nginx Proxy Manager

- Select **Save** to complete the proxy host setup.

Your reverse proxy for the NPM, s Management UI service is now in place. Navigate to your chosen domain to see it in action.

Do the same process to Portainer's Management interface by entering portainer's IP address and port **9443**. Don't forget to set the scheme to **HTTPS**.

This process is rinse-and-repeat for most of the App/containers you will deploy. Below we will make available the docker-compose.yml files for all the stacks.

Use Bookstack's docker-compose.yml supplied below, to deploy bookstack or any other container or stack.

Docker-compose.yml Files

NGINX Proxy Manager

```
version: '3.8'

services:
  app:
    image: 'jc21/nginx-proxy-manager:latest'
    restart: unless-stopped
    ports:
      # These ports are in format <host-port>:<container-port>
      - '80:80' # Public HTTP Port
      - '443:443' # Public HTTPS Port
      - '81:81' # Admin Web Port
      # Add any other Stream port you want to expose
      # - '21:21' # FTP
    environment:
      # Mysql/Maria connection parameters:
      DB_MYSQL_HOST: "db"
      DB_MYSQL_PORT: 3306
      DB_MYSQL_USER: "npm"
      DB_MYSQL_PASSWORD: "npm" #change this line
      DB_MYSQL_NAME: "npm"
      # Uncomment this if IPv6 is not enabled on your host
      # DISABLE_IPV6: 'true'
    volumes:
      - ./data:/data
      - ./letsencrypt:/etc/letsencrypt
    depends_on:
      - db
  networks:
```

- net-to-proxy
- npm_default

db:

image: 'jc21/mariadb-aria:latest'

restart: unless-stopped

environment:

MYSQL_ROOT_PASSWORD: 'npm' #change this line

MYSQL_DATABASE: 'npm'

MYSQL_USER: 'npm'

MYSQL_PASSWORD: 'npm' # change this line

volumes:

- ./mysql:/var/lib/mysql

networks:

- npm_default

networks:

net-to-proxy:

external: true

npm_default:

driver: bridge

Portainer

version: "3"

services:

portainer:

image: portainer/portainer-ce:latest

ports:

- 9443:9443

volumes:

- data:/data
- /var/run/docker.sock:/var/run/docker.sock

restart: unless-stopped

networks:

- portainer_net
- net-to-proxy

volumes:

data: ./poertainer_data

networks:

portainer_net:

driver: bridge # This means that, this network is using the host bridge

net-to-proxy:

external: true # This means that, this network was created previously and is external

Bookstack

version: "2"

services:

bookstack:

image: lscr.io/linuxserver/bookstack

container_name: bookstack

environment:

- PUID=1001
- PGID=1001
- APP_URL=https://your_domain.tld
- DB_HOST=bookstack_db
- DB_PORT=3306
- DB_USER=bookstack
- DB_PASS=Change_this_line
- DB_DATABASE=bookstackapp

volumes:

- ./bookstack_app_data:/config

ports:

- 6875:80

restart: unless-stopped

depends_on:

- bookstack_db

networks:

- bookstack_default
- net-to-proxy

bookstack_db:

image: lscr.io/linuxserver/mariadb

container_name: bookstack_db

environment:

- PUID=1001

- PGID=1001
- MYSQL_ROOT_PASSWORD=Change_this_line
- TZ=America/New_York
- MYSQL_DATABASE=bookstackapp
- MYSQL_USER=bookstack
- MYSQL_PASSWORD=Must_match_password

volumes:

- ./bookstack_db_data:/config

restart: unless-stopped

networks:

- bookstack_default

networks:

bookstack_default:

driver: bridge

net-to-proxy:

external: true

How to create and manage Docker networks

Docker allows you to create specific networks and attach containers to them. Here's how to make use of this highly flexible feature.

 or type unknown

Docker is one of the most flexible and user-friendly container systems on the market. Once up to speed on the platform, there's very little you can't do. Of course, the more you learn about Docker, the more you realize there is to learn about Docker. One such instance is Docker networks. Did you know you can actually create networks that offer complete isolation for Docker and then deploy containers on those isolated networks?

Out of the box, Docker creates three networks:

- bridge – An automatically generated network with a subnet and a gateway.
- host – Allows a container to attach to the host's network.
- none – A container-specific network stack that lacks a network interface.

Docker connects to the bridge network by default; this allows deployed containers to be seen on your network. Let's see how we can manage those networks, create a new network, and then deploy a container on our new network.

Viewing networks

To view the current list of Docker networks, issue the command:

```
docker network ls
```

The above command will list out the Docker networks (**Figure A**).

Figure A

image not found or type unknown



You can get more information on a particular network, by issuing the command `docker network inspect NAME` (Where NAME is the name of the network you want to view). If you want to view details on the bridge network, that command would be `docker network inspect bridge`. The output of that command would give you all the information you need about that network (**Figure B**).

Figure B

image not found or type unknown



Creating a new network

I'm going to demonstrate how to create a bridge network and then show you how to deploy a container on that network. We'll create a network called `isolated`. The creation of this network can be achieved with a single command:

```
docker network create --driver bridge isolated
```

The output of that command will be a long string of characters that represents the ID of that newly-created network (**Figure C**).

Figure C

image not found or type unknown



Run the inspect command on that newly created network with the command `docker network inspect isolated` to see that our new network has been automatically given its own subnet and gateway (**Figure D**).

Figure D

image not found or type unknown

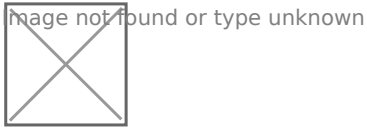


But what if you want to create a network with a specific subnet and gateway? That's possible as well. Let's say you want to create a network with a subnet of `192.168.2.0/24`, a gateway of `192.168.2.10`, and the name `new_subnet`. The command for this would be:

```
docker network create --driver=bridge --subnet=192.168.2.0/24 --gateway=192.168.2.10 new_subnet
```

Once created, inspect the network, with the command *docker network inspect new_subnet* to see the results (**Figure E**).

Figure E



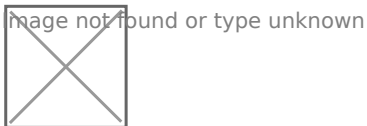
Attaching a container to a network

Let's attach a container to our newly created network. Say you've already pulled down the nginx image and want to launch a container, named `docker-nginx`, attached to the isolated network. To do this, the command would look like:

```
docker run --network=isolated -itd --name=docker-nginx nginx
```

If you now run the command `docker network inspect isolated`, you'll see that the container has been attached (**Figure F**).

Figure F



Any other container you create on this network would be able to automatically connect to one another. So if you create a database container on `isolated`, it would be available for any other container on the same network.

How to Use Docker Compose

Purpose

Learn how to define and manage multiple Docker containers using a simple YAML configuration file with Docker Compose.

1. Install Docker Compose

If you don't already have Docker Compose installed, install it:

bash

```
sudo apt update
sudo apt install docker-compose-plugin -y
```

2. Create a Docker Compose File

Create a directory for your project and add a `docker-compose.yml` file inside it:

bash

```
mkdir my-project
cd my-project
nano docker-compose.yml
```

Example `docker-compose.yml`:

yaml

```
version: '3'

services:
  web:
```

```
image: nginx
```

```
ports:
```

```
- "8080:80"
```

3. Start the Application

Use Docker Compose to bring up your containers:

bash

```
docker compose up -d
```

The `-d` option runs it in the background (detached mode).

4. Stop the Application

When you want to stop and remove all containers defined in the Compose file:

bash

```
docker compose down
```

□ Summary

- **Compose simplifies** multi-container deployments with a single file.
- **Use volumes** inside `docker-compose.yml` to persist your data.
- **Keep your Compose files** under version control (like Git) for easy recovery and updates.

How to Mount NFS Storage for Docker Containers

Purpose

Mount a remote NFS share on a Linux server for use by **Docker containers**, ensuring stable operation, correct permissions, and automatic remounting.

1. Install NFS Client on the Server

```
sudo apt update
sudo apt install nfs-common -y
```

2. Create a Local Mount Directory

Create a local directory where the NFS share will be mounted:

```
sudo mkdir -p /srv/nfs-mount
sudo chown $(whoami):$(whoami) /srv/nfs-mount
```

(You can replace `/srv/nfs-mount` with your preferred path.)

3. Mount the NFS Share (Manual Test)

Example:

```
sudo mount -t nfs4 192.168.100.11:/mnt/hdd-storage/my-nfs-share /srv/nfs-mount
```

- `nfs4`: Use NFS version 4 for better performance and locking.
- `proto=tcp`: Reliable transport.
- `hard`: Wait for server recovery instead of failing immediately.
- `timeo=600`: Timeout setting for NFS.
- `retrans=2`: Retry failed operations twice.
- `sec=sys`: Default authentication security.
- `_netdev`: Only mount after network is ready.

4. Verify That the Mount Worked

```
mount | grep nfs
```

You should see:

```
192.168.100.11:/mnt/hdd-storage/my-nfs-share on /srv/nfs-mount type nfs4 (...)
```

5. Make the Mount Persistent (Auto-Mount on Boot)

Edit your `/etc/fstab` file:

```
sudo nano /etc/fstab
```

Add the following line:

```
192.168.100.11:/mnt/hdd-storage/my-nfs-share /srv/nfs-mount nfs4  
rw,relatime,hard,proto=tcp,timeo=600,retrans=2,sec=sys,_netdev 0 0
```

Save and exit (`Ctrl+O`, `Enter`, `Ctrl+X`).

6. Test the fstab Entry Without Rebooting

```
sudo mount -a
```

No errors = success ☐

7. Using the NFS Mount with Docker

When running your containers, **bind-mount** your NFS storage into the container:

```
docker run -d \  
  --name my-container \  
  -v /srv/nfs-mount:/app/data \  
  my-docker-image
```

This will allow your Docker containers to directly access the NFS storage.

☐ Summary

- **Use NFSv4** (`nfs4`) whenever possible.
 - **Always** include `_netdev` in `/etc/fstab`.
 - **Use `hard` mounts** to protect container file operations during NFS server issues.
 - **Bind-mount** the NFS path inside containers carefully.
 - **Avoid** using NFS for databases unless network latency is very low.
-

☐☐ Useful Commands

- Mount NFS manually: `sudo mount -t nfs4 server:/path /mountpoint`

- Check current NFS mounts: `mount | grep nfs`
 - Reload fstab: `sudo mount -a`
-

☐☐ Additional Notes

- If the NFS server reboots, containers may pause if heavily dependent on storage.
- When using Compose (`docker-compose.yml`), define volumes mapped to NFS mount points.
- For production, consider using **systemd mount units** for even better recovery handling.

How to Mount a Local Folder into a Docker Container

Purpose

Bind-mount a local folder into a Docker container, enabling persistent storage and easy sharing of files between host and container.

1. Create the Local Folder

Create a folder on the host that you want to share:

```
mkdir -p ~/docker-data/myapp
```

2. Set Correct Permissions

Ensure the folder is accessible by Docker containers:

```
chmod 755 ~/docker-data/myapp
```

(Adjust permissions based on your security requirements.)

3. Run a Container with the Folder Mounted

Use the `-v` flag in Docker:

```
docker run -d \  
  --name my-container \  
  -v ~/docker-data/myapp:/app/data \  
  my-docker-image
```

- Left side (`~/docker-data/myapp`): Host path.
 - Right side (`/app/data`): Path inside the container.
-

4. Verify the Mount

Inside the container, the folder should appear:

```
docker exec -it my-container ls /app/data
```

If you create files inside `/app/data`, they will appear inside your host folder too.

□ Summary

- **Bind-mounts** connect host folders to containers.
 - Use absolute paths when mounting folders in production.
 - Ensure correct permissions to avoid container access issues.
-

□□ Useful Commands

- Check container mounts: `docker inspect my-container`
- Restart containers easily: `docker restart my-container`

How to Backup and Restore a Docker Volume

Purpose

Backup and restore Docker volumes easily for disaster recovery or migration between servers.

1. List Your Volumes

First, identify which volume you want to backup:

```
docker volume ls
```

2. Backup a Volume

Create a compressed backup (.tar.gz) of the volume:

```
docker run --rm \
-v my-volume:/volume \
-v $(pwd):/backup \
alpine \
tar czf /backup/my-volume-backup.tar.gz -C /volume .
```

- `my-volume`: The name of the volume you are backing up.
 - `$(pwd)`: Your current directory will store the backup file.
-

3. Restore a Volume

Restore the volume from a backup archive:

```
docker run --rm \  
-v my-volume:/volume \  
-v $(pwd):/backup \  
alpine \  
tar xzf /backup/my-volume-backup.tar.gz -C /volume
```

□ Summary

- **Always test your backups** after creating them.
- Volumes can be moved between servers by copying the .tar.gz files.
- Use `alpine` for lightweight backup/restore operations inside containers.

□□ Useful Commands

- Inspect volume: `docker volume inspect my-volume`
- Delete unused volumes: `docker volume prune`

How to Backup and Restore Docker Volumes

Purpose

Backup and restore Docker volumes to protect persistent container data or migrate storage between servers.

1. Backup a Docker Volume

Use `tar` to archive a Docker volume:

```
docker run --rm \
  -v my_volume:/volume \
  -v $(pwd):/backup \
  alpine \
  tar czf /backup/volume-backup.tar.gz -C /volume .
```

- `my_volume`: Name of the Docker volume.
 - `$(pwd)`: Current directory where the backup will be saved.
 - `volume-backup.tar.gz`: Output backup file.
-

2. Restore a Docker Volume

Use `tar` to extract the backup:

```
docker volume create my_volume
docker run --rm \
  -v my_volume:/volume \
  -v $(pwd):/backup \
```

```
alpine \  
sh -c "cd /volume && tar xzf /backup/volume-backup.tar.gz"
```

- Ensure you restore into a clean, newly created volume to avoid conflicts.
-

□ Summary

- **Always backup volumes** before upgrading or moving containers.
 - Use lightweight images like `alpine` to create archives easily.
-

□□ Useful Commands

- List Docker volumes: `docker volume ls`
- Inspect a volume: `docker volume inspect my_volume`