RustFS Instance

RustFS is an open-source, S3-compatible distributed object storage system built in Rust, designed for high performance and memory safety.

- NFS Share + RustFS Docker Deployment
- TrueNas VM + RustFS

NFS Share + RustFS Docker Deployment

In this tutorial we will explain how you setup your own RustsFS S3 Storage. This will allow you to use S3 storage for self hosted applications that allow or require it. It is yet another way of storing photos and videos but instead of using an NFS share or local storage. Now, ideally you setup a dedicated box with hard drives to run your RustFS instance. However, in a HomeLab that is not always feasible nor necessary. The beauty of RustFS is that it just needs a folder to store all the objects in. Therefor, if you already have an NAS or NAS VM you could create an NFS share and attach it to that folder. This is what we will do in this tutorial

Setup the NFS Share

1. The first thing we need to do on your TrueNas instance is create a new user: 1001 and group 1001. The reason for this is because RustFS runs on this user and group and hence the objects folder /mnt/rustfs needs to be owned by that user and group. For instructions check out our TrueNas Book

2. Then create a new dataset and attach the user 1001 and group 1001 to it

3. Then setup the NFS share on your TrueNas instance.

4. Sweet, now you setup on TrueNas is set and done. Now, you will enter either your DockerVM or a new dedicated VM to run your RustFS instance on. Regardless we first make a new folder

```
mkdir rustfs
```

5. We then make the data directory that will store all the objects with. You can choose any location and name but for this example we will use

```
mkdir /mnt/rustfs
```

6. Next make sure that an NFS-Client is installed and if not it is easy to install with

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

7. Perfect, now attach the NFS share to the /mnt/rustfs folder by adding a new entry in the /etc/fstab section

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

8. Then add the following line

```
IP-Address-TrueNasVM:/mnt/tank/rustfs /mnt/rustfs  nfs  defaults 0 0
```

9. Alrightieee!!! NFS and Storage folder is all setup, then now we can move to the Docker compose file

Docker Compose File

Below the simple docker compose file of RustFS is provided. Where simple means that it is only rustfs and not all extra add-ons to measure the performance and maintain the instance e.g Prometheus etcetera. This file can be obtained at their repository page. Do note that the Github version sets 4 folders whereas we only use one folder called rustfs. This is because we have the redundancy already embedded in the NFS share.

compose.yml

```
services:
  rustfs:
    image: rustfs/rustfs:latest
    container_name: rustfs
    volumes:
      - /mnt/rustfs:/data
      - ./logs:/app/logs
    environment:
      - RUSTFS_VOLUMES=/data
      - RUSTFS_ADDRESS=0.0.0.0:9000
      - RUSTFS_CONSOLE_ADDRESS=0.0.0.0:9001
      - RUSTFS_CONSOLE_ENABLE=true
      # Without this, logs go to stdout and ./logs stays empty
      - RUSTFS_OBS_LOG_DIRECTORY=/app/logs
      - RUSTFS_ACCESS_KEY=${RUSTFS_ACCESS_KEY:?set_a_non_default_access_key}
      - RUSTFS_SECRET_KEY=${RUSTFS_SECRET_KEY:?set_a_non_default_secret_key}
    ports:
      - "9000:9000"
      - "9001:9001"
    healthcheck:
```

```
      test: ["CMD", "sh", "-c", "curl -f http://10.69.4.90:9000/health && curl -f
http://10.69.4.90:9001/rustfs/console/health"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s
    restart: unless-stopped
    deploy:
      resources:
        limits:
          cpus: "2.0"
          memory: 4GB
```

Things to Note in this compose file

> **1.** The location of your data folder. In this example it is located at **/mnt/rustfs** which is again mounted to the NFS share. Change this if your object storage folder is located somewhere else

> **2.** You can choose the port you want to host your RustFS instance on, by default this is 9001

> **3**. The second things is to have localhost in he healthcheck test changed to your VM's IP address e.g 10.50.20.20

That's all for the compose file

Environment Variables File

Then the compose file is pretty simple as it only requires two entries

.env

```
RUSTFS_ACCESS_KEY=A37045B70F04225A2EC24718
RUSTFS_SECRET_KEY=wIc1icWwzBoUoJlO752fW/13/aq3EHUfXj5trcR9QUE=
```

> Access Key can be generated with: **openssl rand -hex 12 | tr 'a-f' 'A-F'**

>

> SECRET_KEY can be generated with:**openssl rand -base64 32**

Default RustFS Region

Yes — the default RustFS region is `us-east-1`, defined as the `RUSTFS_REGION` constant, and you can override it by setting the `RUSTFS_REGION` environment variable (or the `--region` CLI flag / `config.rustfs.region` Helm value). app.rs:289-292 config_struct.rs:233-234

Domain Name

If you have an application that needs to be accessible over the internet or want other people make use of your S3 instance then you need to add a public domain name to your instance. In the case of Caddy on your OPNSense router that is simply adding the domain name and point it to your VM where RustFS is running on and the port your have set, which by default is 9001

Deploy

Then you can simply deploy the compose stack with

```
docker compose up -d
```

After you have deployed it and your services are healthy then you should be able to access it via the domain name you have set for your RustFS instance.

First Time Login

The first time you enter the RustFS Dashboard you need to login with the ACCESS and SECRET keys you have set in your **.env** file

- Account: ACCESS_KEY
- Key: SECRET_KEY

Then after that we directly will create a new user account that is not the root account

1. To do this go to Users > + Add users
2. Then create a username and password
3. Make sure you select the consoleAdmin role s.t your user has administrator permissions

```
dpasn@sandbox:~$ sudo chown dpasn:dpasn -R /mnt
dpasn@sandbox:~$ sudo chmod 775 -R /mnt
dpasn@sandbox:~$ mkdir /mnt/rustfs
```

Make sure NFS Client is installed with

TrueNas VM + RustFS

Suggested setup

1. TrueNAS VM: Pass the disks or, preferably, their controller through to TrueNAS so ZFS sees the actual drives. Don’t create the pool from virtual disks if you can avoid it; TrueNAS warns that virtualized storage for regular or critical data is not a supported setup and may risk corruption. truenas.com
2. Make separate datasets: For example, `pool/apps-nfs` and `pool/rustfs-data`. Export them over NFS, restrict the exports to your app/RustFS VM addresses, and keep permissions deliberate.
3. RustFS VM: Keep its OS and VM disk on Proxmox’s local storage; mount `pool/rustfs-data` inside the VM and configure RustFS to use that mount as its data directory. RustFS supports single-node operation and an S3-compatible API.
4. Apps: Point Ente and PeerTube to RustFS’s S3 endpoint. Make that endpoint reachable from both the app/server side and the clients that upload directly—Ente’s presigned uploads require both to reach the same address. PeerTube’s S3 setup also requires attention to public access and CORS for video delivery.
5. Other apps: Mount `pool/apps-nfs` directly from TrueNAS where a regular shared filesystem is appropriate.

I’d avoid using the same NFS export for Proxmox VM disks and for application/object data. If you do want NAS-backed VM disks, make a separate dataset/export for that. NFS can work for Proxmox VM storage, but it adds network and NAS availability as dependencies for the VMs; keeping their disks local is simpler for this design.

Important tradeoff: the RustFS data is still on the mirror, so it is protected against one disk failing, but the mirror is not a backup. Take separate backups of important photos and videos. Also, with TrueNAS and RustFS as separate VMs, RustFS depends on TrueNAS being up and its NFS mount being available. A single mirrored pair does not provide storage high availability.