Docker

Notes on running containers with Docker Engine, Docker Compose and Podman, plus self-hosted apps such as Pi-hole and RustDesk.

- RedHat - Install Docker Engine on RHEL 9
- PiHole - Synology Configuration
- .gitignore - .gitIgnore File for Docker Compose Files
- RustDesk - pfSense and and HAProxy
- PODMAN - Install Podman on Linux Redhat
- Docker - Linux Docker Compose Wrapper

RedHat - Install Docker Engine on RHEL 9

> Last verified: October 2026 for **RHEL 9** against Docker's Install Docker Engine on RHEL guide, Linux post-installation steps, and packet-filtering / firewalld notes (see **Sources**).

This installs **Docker Engine Community (Docker CE)** from Docker's official RHEL repository: the `dockerd` daemon, the `docker` CLI, `containerd`, Buildx, and the Compose v2 plugin. It is **not** Docker Desktop, and it is **not** Red Hat's Podman stack.

Red Hat ships **Podman** as the supported engine on RHEL 9. Use this page when you specifically need Docker Engine. If you want Podman instead, use the Podman page in the Docker book.

Prerequisites

- RHEL 9.x (64-bit). Docker documents RHEL 8, 9 and 10. This page is written for RHEL 9.
- An active Red Hat subscription so `dnf` can install base dependencies.
- A sudo-capable account.
- Outbound HTTPS to `download.docker.com` (and the usual RHEL repos).

Rocky Linux and AlmaLinux are RHEL rebuilds. Docker's documented repo for this procedure is `download.docker.com/linux/rhel`. Docker does not list Rocky/Alma as tested platforms on the RHEL install page.

1) Conflicts with Podman

Docker's packages conflict with the distribution's unofficial Docker packages **and with Podman / runc**. The uninstall step below is what Docker documents. **Do not run it on a host that must keep Podman.** Pick one engine per host.

```
sudo dnf remove docker \
  docker-client \
  docker-client-latest \
  docker-common \
  docker-latest \
  docker-latest-logrotate \
```

```
    docker-logrotate \
    docker-engine \
    podman \
    runc
```

`dnf` may report that none of those packages are installed. Images, containers, volumes and networks under `/var/lib/docker/` are not removed by this command.

2) Add Docker's RHEL repository

```
sudo dnf -y install dnf-plugins-core
sudo dnf config-manager --add-repo https://download.docker.com/linux/rhel/docker-ce.repo
```

Confirm the repo is present:

```
dnf repolist docker-ce-stable
```

3) Install Docker Engine

Latest packages:

```
sudo dnf install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

If DNF asks you to accept the GPG key, check the fingerprint matches Docker's published key before you accept:

```
060A 61C5 1B55 8A7F 742B 77AA C52F EB6B 621E 9F35
```

This installs Docker but does **not** start it. It creates a `docker` group and does not add any users to that group.

To pin a version, list what the repo currently has, then install the fully qualified NEVRA (name + epoch/version from the second column):

```
dnf list docker-ce --showduplicates | sort -r
sudo dnf install docker-ce-<VERSION_STRING> docker-ce-cli-<VERSION_STRING> containerd.io docker-buildx-plugin docker-compose-plugin
```

Example NEVRA from Docker's docs at verification time: `docker-ce-3:29.8.2-1.el9`. The list on your host will change as Docker publishes updates.

4) Start and enable the service

```
sudo systemctl enable --now docker
sudo systemctl enable containerd.service
sudo systemctl status docker --no-pager
```

If you do not want Docker at boot, use `sudo systemctl start docker` instead of `enable --now`.

5) Verify

```
sudo docker version
sudo docker compose version
sudo docker run --rm hello-world
```

`hello-world` pulls a test image, runs a container, prints a confirmation message, and exits. Use `sudo` until you add a user to the `docker` group.

6) Configure Docker's default address pools

The address pools control which subnets Docker hands out to new bridge networks. Each network gets a /24, so Docker does not run out of address space. Create or edit the daemon config:

```
sudo vi /etc/docker/daemon.json
```

Add:

```
{
  "default-address-pools": [
    {
      "base": "172.17.0.0/12",
      "size": 24
    },
    {
      "base": "192.168.0.0/16",
      "size": 24
    },
    {
```

```
    "base": "10.100.0.0/16",
    "size": 24
   }
  ]
}
```

> If `/etc/docker/daemon.json` already exists, merge the `default-address-pools` key into it instead of overwriting the file. Pick pools that do not overlap your LAN or VPN ranges.

Restart Docker and check that a new network gets a /24 from the pools:

```
sudo systemctl restart docker
sudo docker network create testnet && sudo docker network inspect testnet | grep Subnet && sudo docker network rm testnet
```

The `Subnet` line should show a /24 from one of the pools. `sudo docker info` also lists them under `Default Address Pools`.

7) Optional: run docker without sudo

> The `docker` group is root-equivalent. Anyone in that group can control the daemon and, from there, the host. Do not add accounts you would not give root. For stronger isolation, use rootless mode (see Sources) or keep using `sudo docker`.

```
sudo usermod -aG docker YOUR_USER
```

Log out and back in (or reboot a VM) so the group is applied. Then:

```
docker run --rm hello-world
```

If you used `sudo docker` earlier and see a permission error on `~/.docker/config.json`:

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

8) firewalld and SELinux (leave them on)

Keep **SELinux enforcing**. Do not set it to permissive for this install.

Keep **firewalld** enabled. When Docker's iptables integration is on (the default), Docker creates a firewalld zone named `docker` with target `ACCEPT`, puts bridge interfaces such as `docker0` in that zone, and adds a `docker-forwarding` policy from ANY zone to the `docker` zone. You do not need to disable firewalld for published ports on the Docker host.

```
sudo getenforce
sudo systemctl is-active firewalld
sudo firewall-cmd --get-active-zones
```

Do not open the Docker socket on TCP (2375/2376) unless you have a TLS plan. This page leaves the daemon on the default Unix socket only.

9) Upgrade

With the repo in place, upgrade the same packages:

```
sudo dnf upgrade docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
sudo systemctl restart docker
```

10) Uninstall

```
sudo dnf remove docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin docker-ce-rootless-extras
```

That does not delete images, containers, volumes or custom config. Only if you intend to wipe local Docker data:

```
sudo rm -rf /var/lib/docker
sudo rm -rf /var/lib/containerd
```

Remove any files you edited under `/etc/docker/` yourself.

The `get.docker.com` convenience script is not recommended on production RHEL hosts. Use the repository steps above.

Sources

- https://docs.docker.com/engine/install/rhel/
- https://docs.docker.com/engine/install/linux-postinstall/
- https://docs.docker.com/engine/network/packet-filtering-firewalls/

- https://docs.docker.com/engine/security/rootless/
- https://docs.docker.com/engine/install/security/#docker-daemon-attack-surface

PiHole - Synology Configuration

This is to create a MAC vlan on you synology to bridge interfaces

You will need to create the following folders from the main docker folder.

```
mkdir -p /volume1/docker/pihole
mkdir -p /volume1/docker/pihole/pihole
mkdir -p /volume1/docker/pihole/dnsmasq.d
```

My testing vlan uses 192.168.10.0/24

```
sudo docker network create -d macvlan -o parent=eth3 --subnet=192.168.10.0/24 --gateway=192.168.10.1 --ip-range=192.168.10.44/32 ph_network
```

YAML Configurations

```
services:
  pihole:
    image: pihole/pihole:latest
    container_name: pihole
    hostname: pihole
    security_opt:
      - no-new-privileges=false
    networks:
      - ph_network
      - ph_bridge
    environment:
      PIHOLE_GID: 100
      PIHOLE_UID: 1024
      DNSMASQ_USER: root
      TZ: 'America/Montreal'
      FTLCONF_webserver_api_password: '[Your_Password_Here]'
      #FTLCONF_webserver_port: 8080
      FTLCONF_dns_listeningMode: all
```

```
    volumes:
      - /volume1/docker/pihole/pihole:/etc/pihole
      - /volume1/docker/pihole/dnsmasq.d:/etc/dnsmasq.d
      - /etc/localtime:/etc/localtime:ro
    cap_add:
      - NET_ADMIN
      - SYS_TIME
      - SYS_NICE
    restart: on-failure:5
networks:
    ph_bridge:
      driver: bridge
      ipam:
        config:
          - subnet: 192.168.100.0/24
            gateway: 192.168.100.1
            ip_range: 192.168.100.2/32
    ph_network:
      name: ph_network
      external: true
```

Accessing the pi-hole remotely

```
https://192.168.10.44/admin
```

If you are 7.1 DSM you will need to use this command

```
docker-compose -f /volume1/docker/pihole/compose.yaml up -d
```

Note: Pi-hole v6 sets the listening mode with `FTLCONF_dns_listeningMode`; the v5 `DNSMASQ_LISTENING` variable is not used.

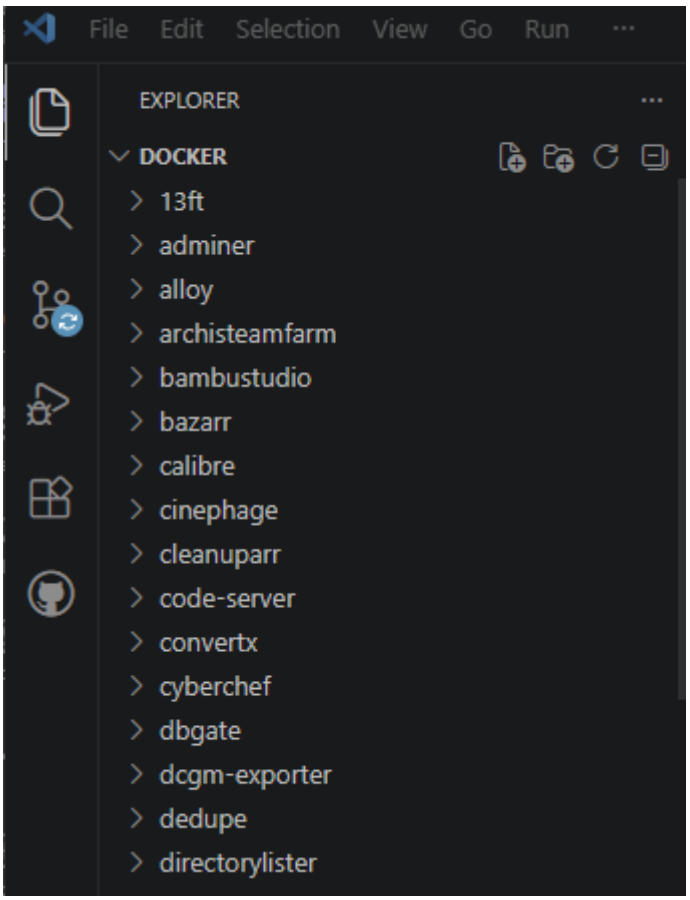
Last verified: October 2026

Sources:

- https://docs.pi-hole.net/docker/configuration/

.gitignore - .gitIgnore File for Docker Compose Files

Use Case TrueNAS shared SMB with a folder dedicated to yaml files.



This is it to be used if you have a SMB drive and point Visual Code or Code Server to all of your docker folders.

The .`gitignore` file setup

```
# Ignore everything by default
*

# Allow directories to be traversed
!*/

# Include YAML files
```

```
!*.yaml
!*.yml
!.env
!env.*

# Include .gitignore and .gitattributes files
!.gitignore
!.gitattributes
```

RustDesk - pfSense and and HAProxy

You can configure self-hosted RustDesk (hbbs ID/rendezvous server + hbbr relay server) behind your pfSense + HAProxy Docker setup using `hostname.domain.com` as the external ID server address. However, there are important caveats due to how RustDesk works.

Important Limitations with HAProxy for RustDesk

RustDesk uses a **custom protocol** (not HTTP/HTTPS) with:

- **TCP ports**: 21115 (hbbs signaling), 21116 (hbbs), 21117 (hbbr relay)
- **UDP port**: 21116 (critical for NAT traversal / hole punching)
- Optional WebSocket ports 21118/21119 (only needed for the web client)

HAProxy (even in TCP mode) is not ideal for the core RustDesk traffic:

- It cannot properly proxy UDP 21116 (HAProxy's UDP support is very limited and not suitable here).
- Even in TCP mode, the real client IP is hidden (hbbs sees the HAProxy container's IP), which breaks NAT hole-punching and forces everything through the relay.
- Domain-based routing (`hostname.domain.com`) doesn't help because RustDesk clients connect directly to the resolved IP + specific ports — there is no Host header or SNI to route on.

Recommended approach: Use **pfSense Port Forward / NAT rules** for the RustDesk ports (direct to your RustDesk Docker host/container). Keep HAProxy for your HTTP/HTTPS services only. This is the most reliable way and what most people successfully use with pfSense + RustDesk.

If you *must* route everything through HAProxy Docker, it's possible for the TCP ports only (see advanced section at the end), but UDP will still need a direct forward and performance/reliability will suffer.

Step-by-Step Recommended Setup (Direct pfSense NAT + Domain)

1. **Set up DNS**
   - Create an A record (and AAAA if IPv6) for `hostname.domain.com` pointing to your **pfSense WAN public IP**.
   - No special subdomains or paths needed.
2. **Run RustDesk Server in Docker** (on your internal host/VM/LXC behind pfSense) You can also choose to use the official Docker Compose (download `oss.yml` from RustDesk or use this standard one):

```
services:
  hbbs: # ID / signaling server
    container_name: hbbs
    image: rustdesk/rustdesk-server:latest
    command: hbbs -r {REMOTE HOSTNAME}:21117
    logging:
      driver: json-file
      options:
        max-file: ${DOCKERLOGGING_MAXFILE:-10}
        max-size: ${DOCKERLOGGING_MAXSIZE:-200k}
    ports:
      - "21115:21115"
      - "21116:21116"
      - "21116:21116/udp"
      - "21118:21118"
    volumes:
      - ${DOCKERFOLDER}/rustdesk/data:/root
    depends_on:
      - hbbr
    restart: unless-stopped
  hbbr: # Relay server
    container_name: hbbr
    image: rustdesk/rustdesk-server:latest
    command: hbbr
    logging:
      driver: json-file
      options:
        max-file: ${DOCKERLOGGING_MAXFILE:-10}
```

```
    max-size: ${DOCKERLOGGING_MAXSIZE:-200k}
  ports:
    - "21117:21117"
    - "21119:21119"
  volumes:
    - ${DOCKERFOLDER}/rustdesk/data:/root
  restart: unless-stopped
```

- Run with `docker compose up -d`.
- The `./data` folder will contain your `encryption` key pair (`id_ed25519.pub`).
- Check logs: `docker logs hbbs` → look for the **public key** (you'll need this in clients).

3. **pfSense Configuration (Port Forwards + Firewall Rules)** Go to **Firewall → NAT → Port Forward** and create these rules (all source = WAN, destination = your RustDesk Docker host's internal IP):

Protocol	Destination Port	Forward to IP	Forward to Port	Notes
TCP	21115	RustDesk Docker IP	21115	hbbs signaling
TCP	21116	RustDesk Docker IP	21116	hbbs
UDP	21116	RustDesk Docker IP	21116	Critical for hole punching
TCP	21117	RustDesk Docker IP	21117	hbbr relay

   - Set **Source port range** to *any* (RustDesk clients use random source ports).
   - Auto-create the associated firewall rules (or manually create matching WAN rules under **Firewall → Rules → WAN** allowing the same ports).

   Optional but recommended:
   - Enable **System → Advanced → Firewall & NAT → NAT Reflection** (Pure NAT mode) if you want LAN clients to reach the server via the public domain.
4. **Client Configuration** On every RustDesk client (Windows/macOS/Linux/Android/iOS):
   - Go to **Settings → ID/Relay Server**
   - **ID Server**: hostname.domain.com (or `hostname.domain.com:21115` if you changed the port)
   - **Relay Server**: leave blank (hbbs will tell clients to use `hostname.domain.com:21117`)
   - **Key**: paste the public key from your `./data/id_ed25519.pub` file
5. **Test**
   - From outside your network, connect using an ID from one of your machines.
   - Check connection type in the session (should be "Direct" most of the time; falls back to "Relay" if needed).
   - Monitor logs: `docker logs hbbs` and `docker logs hbbr`.

Advanced: Forcing Everything Through HAProxy Docker (Not Recommended)

If you really want HAProxy as the single entry point:

- Forward **all** the ports above from pfSense WAN → your **HAProxy Docker container IP** (instead of directly to RustDesk).
- In your HAProxy config (or pfSense HAProxy package if you're using that), add TCP-mode frontends/backends for each TCP port:

```
frontend rustdesk_hbbs_21115
    bind :21115
    mode tcp
    default_backend rustdesk_hbbs

backend rustdesk_hbbs
    mode tcp
    server hbbs rustdesk-docker-ip:21115

# Repeat for 21116 (TCP only), 21117, etc.
```

- **UDP 21116** → still do a direct pfSense NAT forward to the RustDesk container (bypassing HAProxy).
- You will likely have NAT traversal issues because of the hidden client IP.

For the RustDesk **Web Client** only (if you use it), you can proxy the WebSocket ports (21118/21119) through HAProxy in HTTP mode with proper `Upgrade` headers — similar to the official Nginx example in the RustDesk docs.

This direct pfSense NAT + domain setup is what works reliably for almost everyone running RustDesk behind pfSense. Let me know if you hit any specific error logs or need the exact Docker Compose / HAProxy snippets adjusted for your environment!

PODMAN - Install Podman on Linux Redhat

Production-Ready Step-by-Step Guide Rootless Podman + Quadlet on Red Hat Enterprise Linux 10

This guide is designed for a production environment as assums a fresh install of Redhat. It uses the modern **Quadlet** method (recommended) and follows Red Hat best practices for RHEL 10.

---

1. System Requirements (Production)

Resource	Minimum	Recommended (Production)	Our Systems
CPU	2 cores	4+ cores	8 cores
RAM	4 GB	8–16 GB+	32 GB
Disk	40 GB free	100 GB+ (SSD preferred)	350 GB
cgroup	cgroup v2	cgroup v2 (required)	
SELinux	Enforcing	Enforcing	

Check cgroup version:

```
podman info --format '{{.Host.CgroupVersion}}'
```

Install Needed Tools:

Epel Release

```
subscription-manager repos --enable codeready-builder-for-rhel-10-$(arch)-rpms
dnf install https://dl.fedoraproject.org/pub/epel/epel-release-latest-10.noarch.rpm -y
```

After Epel installation rerun the upgrade to update if any are needed

```
dnf upgrade -y
```

Toolset

```
dnf install bind-utils bzip2 cups cifs-utils enscript ftp gdb ghostscript krb5-workstation ksh lftp lrzsz lsof libnsl lzop plocate mutt ncompress net-tools net-snmp net-snmp-utils net-tools nfs-utils nmap nvme-cli openldap-clients openssh-clients psmisc realmd rsync samba-client strace sysstat tcpdump telnet telnet-server tmux unix2dos vim vim-enhanced vsftpd wget xfsdump vsftpd htop mc rsyslog rsyslog-doc postfix dbus-daemon s-nail dovecot cyrus-sasl cyrus-sasl-lib cyrus-sasl-plain tree figlet toilet coreutils -y
```

---

2. Prepare the System

2.1 Register and Update RHEL 10

```
sudo dnf update -y
sudo reboot
```

2.2 Install Container Tools

```
sudo dnf install -y container-tools
```

Optional (Docker CLI compatibility):

```
sudo dnf install -y podman-docker
```

Verify:

```
podman --version
podman info
```

---

3. Create a Dedicated Service User (Best Practice)

For production, avoid running services under a regular interactive user.

```
# Create a system user for containers
sudo useradd -r -m -d /home/podman -s /bin/bash podman
```

```
# Set a strong password or disable password login
sudo passwd podman          # or lock the account later
```

Grant subordinate UIDs/GIDs (required for rootless):

```
sudo usermod --add-subuids 100000-165535 --add-subgids 100000-165535 podman
```

---

4. Enable Lingering (Critical for Production)

This allows the user services to run even when the user is not logged in.

```
sudo loginctl enable-linger podman
```

Verify:

```
loginctl show-user podman | grep Linger
# Linger=yes
```

---

5. Configure Rootless Environment

Switch to the service user:

```
# systemctl --user needs a real login session; sudo -u / su do not create one
sudo dnf install -y systemd-container
sudo machinectl shell podman@
```

Create required directories:

```
mkdir -p ~/.config/containers/systemd
mkdir -p ~/.config/containers
mkdir -p ~/.local/share/containers
```

Optional but Recommended: Storage Configuration

```
cat > ~/.config/containers/storage.conf << 'EOF'
[storage]
driver = "overlay"
# Rootless defaults: graphroot = $HOME/.local/share/containers/storage
# and runroot = $XDG_RUNTIME_DIR/containers. Set them here only to use other paths.
# Native overlay works rootless on current RHEL kernels; fuse-overlayfs is not needed.
EOF
```

Optional: Registries Configuration

```
cat > ~/.config/containers/registries.conf << 'EOF'
unqualified-search-registries = ["registry.access.redhat.com", "registry.redhat.io", "docker.io", "quay.io"]

[[registry]]
location = "docker.io"
insecure = false
EOF
```

---

6. Create Your First Production Quadlet

A) Example: Nginx reverse proxy / web service

```
cat > ~/.config/containers/systemd/nginx.container << 'EOF'
[Unit]
Description=Nginx Web Server (Production)
After=network-online.target
Wants=network-online.target

[Container]
Image=docker.io/library/nginx:alpine
ContainerName=nginx
PublishPort=8080:80
Volume=nginx-data.volume:/usr/share/nginx/html:Z
Volume=nginx-conf.volume:/etc/nginx/conf.d:Z
AutoUpdate=registry
Environment=TZ=America/New_York
PodmanArgs=--memory=512m --cpus=1.0 --memory-swap=512m
```

```
[Service]
Restart=always
TimeoutStartSec=300

[Install]
WantedBy=default.target
EOF
```

A) Create supporting volumes:

```
cat > ~/.config/containers/systemd/nginx-data.volume << 'EOF'
[Volume]
VolumeName=nginx-data
EOF

cat > ~/.config/containers/systemd/nginx-conf.volume << 'EOF'
[Volume]
VolumeName=nginx-conf
EOF
```

B) Another Example

```
cat > ~/.config/containers/systemd/myapp.container << 'EOF'
[Unit]
Description=My Application (Production)
After=network-online.target
Wants=network-online.target

[Container]
Image=your-registry/your-app:latest
ContainerName=myapp
PublishPort=8080:8080
Volume=myapp-data.volume:/data:Z
AutoUpdate=registry
Environment=TZ=America/New_York
PodmanArgs=--memory=1g --cpus=1.5 --memory-swap=1g

[Service]
Restart=always
```

```
TimeoutStartSec=300

[Install]
WantedBy=default.target
EOF
```

B) Create supporting volume:

```
cat > ~/.config/containers/systemd/myapp-data.volume << 'EOF'
[Volume]
VolumeName=myapp-data
EOF
```

---

7. Activate the Service

A) Example for Nginx

```
# Reload systemd user units
systemctl --user daemon-reload

# Start and enable
systemctl --user enable --now nginx.service

# Check status
systemctl --user status nginx.service
podman ps
```

View logs:

```
journalctl --user -u nginx.service -f
```

B) Second Example

```
systemctl --user daemon-reload
systemctl --user enable --now myapp.service
```

```
# Verify
systemctl --user status myapp.service
podman ps
```

View logs:

```
journalctl --user -u myapp.service -f
```

---

8. Configure pfSense HAProxy

In pfSense HAProxy:

1. **Backend**
   - Mode: http
   - Server: IP of your RHEL 10 host
   - Port: 8080 (or whatever you published)
   - Health check: recommended
2. **Frontend**
   - Bind to WAN / desired interface
   - TLS offloading (recommended)
   - ACL based on Host header (e.g. hdr(host) -i app.yourdomain.com)
   - Use backend created above
3. **Firewall Rules**
   - Allow traffic from HAProxy (or LAN) to the RHEL host on the published port(s) only.

Firewall example:

```
sudo firewall-cmd --permanent --add-port=8080/tcp
sudo firewall-cmd --reload
```

---

9. Production Hardening Checklist

Item	Command / Action	Status

SELinux	Keep enforcing	☐
Linger enabled	loginctl show-user podman	☐
Resource limits	Set in Quadlet (--memory, --cpus)	☐
Auto-update	AutoUpdate=registry in Quadlet	☐
Firewall	Open only required ports	☐
Log rotation	Configure journald or ship logs	☐
Image scanning	Use an external image scanner (Podman has no built-in scan command)	☐
Backup volumes	Backup ~/.local/share/containers	☐
Non-interactive user	Disable password / use key-based access only	☐
Log management	journalctl --user or forward logs	

Enable auto-update timer (as podman user):

```
systemctl --user enable --now podman-auto-update.timer
```

---

9. Useful Management Commands

A) Example for Nginx

```
# List all user services
systemctl --user list-units --type=service

# Restart a service
systemctl --user restart nginx.service

# Stop a service
systemctl --user stop nginx.service

# View resource usage
podman stats

# Update all containers with AutoUpdate
```

```
podman auto-update
```

B) Another Example

```
# Service management
systemctl --user status myapp.service
systemctl --user restart myapp.service
systemctl --user stop myapp.service

# Container status & resources
podman ps
podman stats

# Logs
journalctl --user -u myapp.service -f

# Manual update
podman auto-update
```

---

10. Optional: Auto-Update Timer

Enable Podman’s auto-update timer (as the podman user):

Bash

```
systemctl --user enable --now podman-auto-update.timer
```

Summary of Key Directories (Rootless)

Purpose	Path
Quadlet files	~/.config/containers/systemd/
Container storage	~/.local/share/containers/storage/
User systemd units	~/.config/systemd/user/
Configuration	~/.config/containers/

Summary

- HAProxy on pfSense = edge reverse proxy / TLS / routing
- RHEL 10 + rootless Podman + Quadlet = application runtime
- No Nginx needed unless the application itself requires it

Last verified: October 2026

Sources:

- https://docs.podman.io/en/latest/markdown/podman-systemd.unit.5.html
- https://github.com/containers/storage/blob/main/docs/containers-storage.conf.5.md

Docker - Linux Docker Compose Wrapper

Docker Compose Wrapper (dc)

A small wrapper around docker compose that automatically loads shared environment files for all stacks.

Why

Docker Compose only interpolates ${VARIABLES} from:

- The shell environment, or
- A local .env file, or
- Files passed with --env-file

It does **not** use the env_file: directive for interpolation.
This wrapper always passes the shared files so every stack gets consistent variable substitution.

Shared Files

/mnt/volume1/docker/stacks/_shared/.env
/mnt/volume1/docker/stacks/_shared/.secret

Install

1. Create the wrapper:

Bash

```
sudo tee /usr/local/bin/dc > /dev/null << 'EOF'
#!/bin/bash
exec docker compose \
  --env-file /mnt/volume1/docker/stacks/_shared/.env \
  --env-file /mnt/volume1/docker/stacks/_shared/.secret \
  "$@"
EOF
```

2. Make it executable:

Bash

```
sudo chmod +x /usr/local/bin/dc
```

3. Verify:

Bash

```
which dc
dc version
```

Usage

Use dc exactly like docker compose.
All subcommands work.

Bash

```
dc up -d
dc logs
dc logs -f
dc ps
dc restart <service>
dc down
dc config
```

Notes

- Run dc from the directory that contains the docker-compose.yml (same as normal Compose).
- Later --env-file overrides earlier ones if keys collide.
- The wrapper does not change how env_file: works inside containers — it only handles interpolation for the Compose file itself.
- To see the fully resolved config before starting anything:

Bash

```
dc config
```

Updating

Edit the script if the shared file paths change:

Bash

```
sudo nano /usr/local/bin/dc
```