ADMIN:DOCS

Install Portainer on Ubuntu 24.04 LTS: manage Docker containers via a web GUI

Hands-on guide to Portainer CE on Ubuntu 24.04 LTS: Docker Engine setup, volume persistence, SSL/TLS, stacks, remote agents, registries and system maintenance.

Category: Linux beginners **Published:** 13.04.2025 **Last reviewed:** 08.09.2026

Managing Docker containers on the command line with `docker run` and the Compose plugin is the proven standard in daily Linux work. On servers that run dozens of microservices, isolated bridge networks, named volumes and periodic image updates in parallel, pure terminal administration hits visibility limits: crash loops have to be analyzed through nested log commands, resource bottlenecks only show up at outages, and orphaned volumes quietly occupy gigabytes of disk space.

Portainer Community Edition (CE) closes this operational gap. As a lightweight, web-based management platform, Portainer docks directly onto the local Docker Engine or onto Portainer agents running remotely. Through a tidy graphical interface you control container life cycles, analyze streaming logs and resource graphs in real time, open interactive shells directly in the browser, deploy multi-container stacks and clean unused artefacts.

On a server with **Ubuntu 24.04 LTS ("Noble Numbat")**, however, running Portainer CE needs a clear security and architecture concept: access to the Docker socket must be controlled, network exposures must be strictly limited, and in production the web interface should sit behind a hardened Nginx reverse proxy with an official TLS certificate.

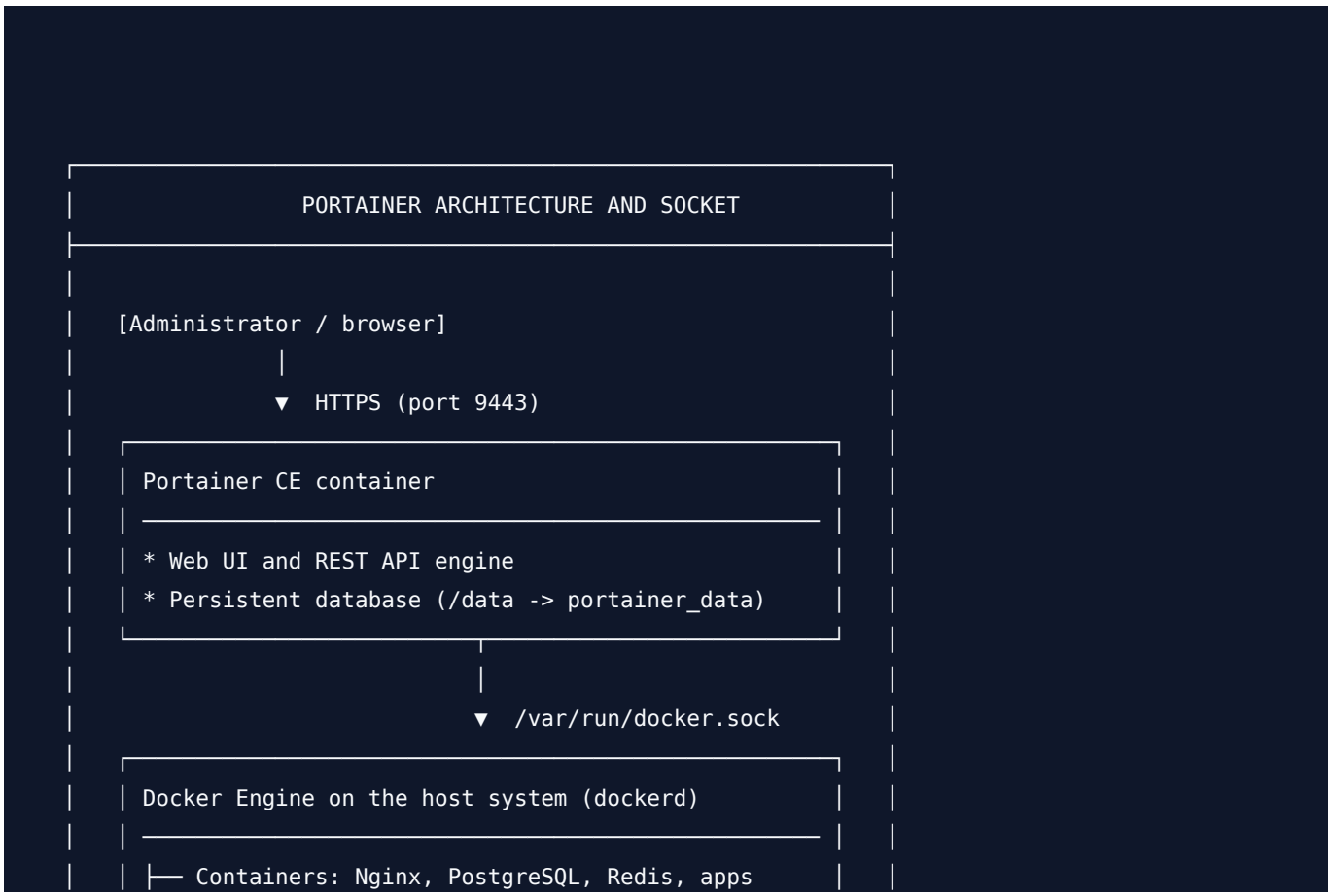
Source: https://admindocs.de/en/linux-beginners/install-portainer-on-ubuntu-24-04-lts

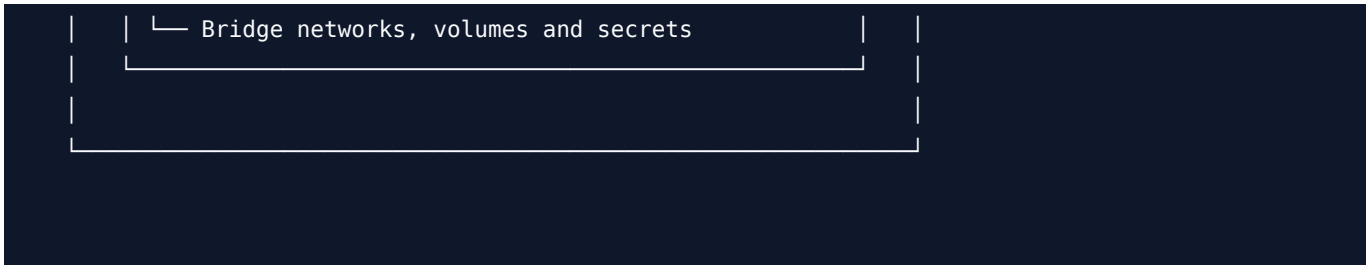
> **Prerequisites:** A server with Ubuntu 24.04 LTS and root or `sudo` rights. If the Docker Engine is not set up yet, we start directly with a clean installation of the official Docker repository. Further containerization fundamentals are in our guide to installing and using Docker on Ubuntu.

Architecture and how Portainer works

How Portainer talks to the Docker Engine

Portainer is not installed as a classic systemd service in the host filesystem, but itself runs as an isolated Docker container. To steer the host daemon and start, stop or monitor containers, the Docker daemon UNIX socket (`/var/run/docker.sock`) is passed into the Portainer container as a volume:





Through this socket mount, Portainer's Go-based backend engine talks directly over HTTP-over-UNIX-socket to the Docker REST API. There is no virtualization overhead, and actions in the web UI are handed to the host daemon without delay.

Portainer Community Edition (CE) vs. Business Edition (BE)

Portainer is maintained in two editions:

Feature	Portainer Community Edition (CE)	Portainer Business Edition (BE)
License model	Open source (free of charge)	Commercial (free up to 3 nodes)
Container and stack control	Fully included	Fully included
Docker Compose support	Built in (editor and upload)	Built in plus Git repository autosync
Access control (RBAC)	Basic (admin and standard user)	Granular team roles and namespace rights
Authentication	Local, basic LDAP and OAuth	Full Active Directory, SAML and SSO
Audit logging	Standard container logs	Tamper-evident activity audit log
Support	Community (Discourse / GitHub)	Commercial 24/7 SLA vendor support

For standalone servers, development environments and small to medium production environments, Community Edition (CE) covers all administrative needs without gaps.

Network ports and security considerations

Portainer binds two communication ports by default:

- **Port 9443 (HTTPS):** The primary, encrypted web port for the dashboard and the REST API. Portainer automatically generates a self-signed TLS certificate on first start.
- **Port 8000 (TCP):** An internal tunnel port for the *Edge Agent*. This port is only needed when remote Docker hosts behind firewalls or NAT are connected over the internet.
- **Port 9000 (HTTP):** The historical, unencrypted web port. For security reasons this port is disabled in modern installations and should no longer be exposed.

> ⚠ **Docker socket security risk:** Access to `/var/run/docker.sock` in practice grants **root rights on the host system**. Anyone who may create new containers in the Portainer dashboard can start a container with `-v /:/host` and modify every file of the host operating system. Protect port 9443 with a strong administrator password, restrict access through the firewall to trusted subnets or a VPN, and additionally read our guide to Linux server hardening with FIDO2 and CrowdSec.

Preparation and Docker installation on Ubuntu 24.04 LTS

1. Updating system packages

Before installation we bring the Ubuntu server to the current package state and install the required helper tools:

```
# Refresh package lists and update installed software
sudo apt update && sudo apt full-upgrade -y

# Provide required tools for secure repositories
sudo apt install -y curl ca-certificates
```

2. Adding the official Docker repository with a modern keyring

We add Docker's official upstream repository to benefit from continuous security updates and the current Docker Engine 27+. According to the modern APT standard we store the key directly as an ASCII-armored key (`.asc`):

```
# Create the directory for isolated keyrings
sudo install -m 0755 -d /etc/apt/keyrings

# Download Docker's official GPG key in ASCII format
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o
/etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Register the official Docker package source in APT
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

3. Installing Docker Engine and the Compose plugin

```
# Synchronize package lists with the new Docker source
sudo apt update

# Install Docker Engine, CLI, containerd and the Compose plugin
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-
compose-plugin

# Enable the Docker service and ensure it starts on boot
sudo systemctl enable --now docker
```

Practical example:

We verify the successful installation and query the active Docker version as well as the socket status:

```
# Check the Docker service status
sudo systemctl is-active docker

# Verify installed versions of Engine and Compose
docker --version
docker compose version
```

The output confirms `active` as well as Docker Engine version 27 or higher.

4. Configuring firewall rules (UFW)

If the default firewall `ufw` is active on the Ubuntu server, the HTTPS port must be opened. Best practice is to limit access to your own administration subnet:

```
# Allow access to port 9443 from the trusted management network
sudo ufw allow from 192.168.1.0/24 to any port 9443 proto tcp comment "Portainer Admin
HTTPS"

# If no separate management network exists (e.g. on cloud servers with IP binding):
# sudo ufw allow 9443/tcp comment "Portainer HTTPS Web UI"

# Verify the firewall status
sudo ufw status verbose
```

Step-by-step installation of Portainer CE

1. Creating a persistent Docker volume

Portainer stores its internal database (user accounts, API tokens, endpoint metadata and stack configurations) in `/data`. So that this configuration survives container updates or server reboots, we create a dedicated Docker volume:

```
# Create a named volume for Portainer configuration
sudo docker volume create portainer_data

# Inspect volume properties
sudo docker volume inspect portainer_data
```

2. Starting the Portainer CE container (LTS release 2026)

For production systems Portainer officially recommends the **LTS release branch** (`portainer/portainer-ce:lts`). That guarantees long-term stability and continuous security patches without monthly STS feature changes affecting the setup:

```
sudo docker run -d \
  -p 8000:8000 \
  -p 9443:9443 \
  --name portainer \
  --restart=always \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v portainer_data:/data \
  portainer/portainer-ce:lts
```

Meaning of the parameters set:

- `-d`: Starts the container in the background (*detached mode*).
- `-p 9443:9443`: Forwards incoming HTTPS traffic from host port 9443 to the container's internal port 9443.
- `-p 8000:8000`: Keeps the TCP port ready for optional Portainer Edge Agents.
- `--name portainer`: Assigns a unique container name used to maintain the service.
- `--restart=always`: Ensures Portainer comes up automatically after a server reboot or a daemon crash.
- `-v /var/run/docker.sock:/var/run/docker.sock`: Binds the local Docker socket so Portainer can steer containers on the host.
- `-v portainer_data:/data`: Maps the created persistent volume onto the Portainer database data folder.
- `portainer/portainer-ce:lts`: Pulls the official long-term support image from Docker Hub.

□□ Practical example:

We check the start process and make sure the web server initialized properly:

```
# Check the running container in the process list
sudo docker ps --filter "name=portainer"

# Inspect the last log lines of the container
sudo docker logs -n 15 portainer
```

The logs show the confirmation: `[INFO] [main] [message: Starting Portainer...]` as well as the notice of the active HTTPS server on port 9443.

3. First initialization in the web browser

Open a browser and call your server's IP address:

```
https://<YOUR-SERVER-IP>:9443
```

Because Portainer generates a self-signed certificate on first start, the web browser first reports a security warning (*“Not a trusted connection”*). Confirm the security exception to reach the initialization form.

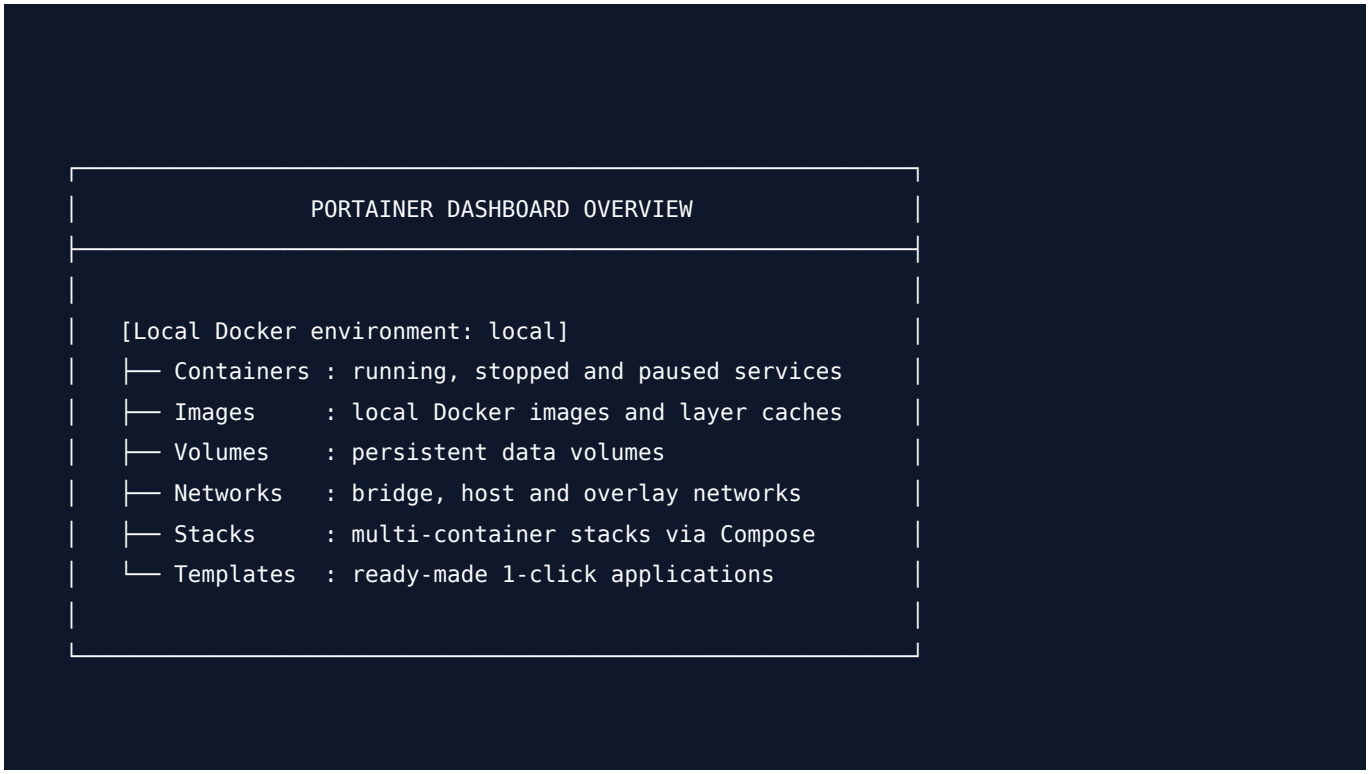
```
┌──────────────────────────────────────────────────────────┐
│                FIRST INITIALIZATION OF THE WEB UI        │
├──────────────────────────────────────────────────────────┤
│                                                          │
│   1. Create the administrator account:                   │
│      * Username: admin (or an individual login)          │
│      * Password: at least 12 characters (high complexity)│
│                                                          │
│   2. 5-minute security timeout:                          │
│      * After start you have exactly 5 minutes            │
│      * On expiry: sudo docker restart portainer          │
│                                                          │
│   3. Environment selection:                              │
│      * Click "Get Started" (local environment)           │
│                                                          │
└──────────────────────────────────────────────────────────┘
```

1. **Create the administrator:** Choose a username and a strong password with at least 12 characters.
2. **Observe the 5-minute protection window:** If no password is set for 5 minutes after the container is first started, Portainer shuts the web UI down automatically for security reasons. If that happens, restart the container with `sudo docker restart portainer`.

3. **Connect the environment:** After account creation, click **Get Started** to add the local Docker Engine (`local`) to the overview automatically.

Tour of the Portainer web interface

After the first login you land on the main dashboard. Click the **local** tile to manage the server's Docker resources:



The core areas in daily administration

- **Containers:**
- Gives a fast overview of status, host port mappings, IP addresses and start times of all containers.
- Action bar: **Start**, **Stop**, **Kill**, **Restart**, **Pause** and **Remove**.
- Built-in diagnostic tools per container:
- **Logs:** Live log streaming with optional timestamp display and search.
- **Inspect:** Raw data and JSON configuration of the container.

Source: https://admindocs.de/en/linux-beginners/install-portainer-on-ubuntu-24-04-lts

- **Stats:** Live graphs for CPU, memory, network and disk I/O use.
- **Console / Exec:** Opens an interactive shell (`/bin/sh` or `/bin/bash`) inside the target container directly in the browser, without an SSH login on the host.
- **Images:**
- Allows targeted downloading (*Pull*) of new images from any registries.
- **Prune** function: Deletes unused intermediate layers (*dangling layers*) and frees disk space.
- **Networks:**
- Manages Docker networks. Here you create isolated custom bridge networks with built-in DNS for clean encapsulation of application landscapes.
- **Volumes:**
- Overview of all named volumes. The **Unused** filter highlights orphaned volumes left behind after deleted containers.
- **App Templates:**
- Preconfigured templates for standard applications (e.g. Redis, Nginx, MariaDB, WordPress) that can be started with one click without writing YAML by hand.

Container and stack management in practice

1. Starting a single container through the web UI

To deploy an Nginx web server, for example:

1. In the menu click **Containers → Add container**.
2. **Name:** `demo-webserver` .
3. **Image:** `nginx:alpine` .
4. **Port mapping:** Choose *publish a new network port* → host `8080` to container `80` (TCP).
5. **Advanced container settings (lower screen area):**

- **Restart policy:** Choose `Unless stopped` .
- **Volumes:** Bind a host directory or a named volume.

1. Click **Deploy the container**. Portainer pulls the image, configures the ports and starts the container.

2. Multi-container applications with stacks (Docker Compose)

In professional server operation, services are rarely run in isolation. A web application typically needs a relational database, caching and a shared internal network. In Portainer this is represented through **stacks**.



> **Compose Specification standard:** Note that in modern Docker Compose files the obsolete attribute `version: '3.8'` is no longer used. According to the current *Compose Specification*, the configuration file starts directly with the `services:` key.

Practical example:

We create a stack consisting of a PostgreSQL 17 database and a web server.

1. In the left menu navigate to **Stacks → Add stack**.
2. Give it the name `production-app`.

3. Paste the following Compose definition into the built-in web editor:

```
services:
  database:
    image: postgres:17-alpine
    container_name: app_postgres
    restart: unless-stopped
    environment:
      POSTGRES_DB: production_db
      POSTGRES_USER: db_user
      POSTGRES_PASSWORD: ExampleSuperSecretPassword2026!
    volumes:
      - db_data:/var/lib/postgresql/data
    networks:
      - backend_network

  webserver:
    image: nginx:alpine
    container_name: app_web
    restart: unless-stopped
    ports:
      - "8080:80"
    depends_on:
      - database
    networks:
      - backend_network

volumes:
  db_data:
    driver: local

networks:
  backend_network:
    driver: bridge
```

1. Click **Deploy the stack**.

Portainer automatically creates the bridge network `production-app_backend_network`, creates the volume and starts both services in the defined dependency. The containers can address each other directly via their service names (`database` and `webserver`) through internal DNS.

Production setup: Nginx reverse proxy with Let's Encrypt TLS

Running Portainer through a self-signed certificate on port 9443 is acceptable for test phases, but error-prone and messy in production. The clean solution is to bind Portainer to the local loopback interface and serve it through an Nginx reverse proxy with an official Let's Encrypt certificate:



> Detailed steps for setting up a web server are in our guide to setting up a LEMP stack on Ubuntu 24.04 LTS (and for newer systems under Ubuntu 26.04 LEMP stack).

Nginx virtual host configuration for Portainer (`/etc/nginx/sites-available/portainer.conf`):

```
server {
    listen 80;
    server_name portainer.example.com;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl http2;
    server_name portainer.example.com;

    ssl_certificate /etc/letsencrypt/live/portainer.example.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/portainer.example.com/privkey.pem;

    # TLS security settings
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers HIGH:!aNULL:!MD5;

    location / {
        proxy_pass https://127.0.0.1:9443;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto https;

        # Important: enable WebSockets for the web terminal and log streaming
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";

        # Disable SSL verification against the container's self-signed certificate
        proxy_ssl_verify off;
    }
```

```
}
```

The `Upgrade` and `Connection` headers are indispensable: if these directives are missing, Portainer's built-in browser console (`docker exec`) refuses the connection.

External registries and multi-node management

Connecting private container registries

To pull images from private repositories (e.g. GitHub Container Registry `ghcr.io`, GitLab or your own Harbor instance):

1. In the left menu navigate to **Settings → Registries → Add registry**.
2. Choose **Custom registry** (or the matching cloud provider).
3. Enter name, registry URL (e.g. `ghcr.io`) and your credentials (username and *Personal Access Token*).
4. Click **Add registry**. From then on Portainer authenticates automatically against this source when pulling images.

Managing several Docker hosts through the Portainer Agent

If several Docker servers must be administered centrally, additional servers do not need a full Portainer dashboard. The extremely resource-light **Portainer Agent** is enough.

On the **remote server (remote node)** run:

```
sudo docker run -d \
  -p 9001:9001 \
  --name portainer_agent \
  --restart=always \
```

```
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v /var/lib/docker/volumes:/var/lib/docker/volumes \
  portainer/agent:lts
```

Then in the **central Portainer dashboard**:

1. Navigate to **Environments → Add environment**.
2. Choose **Docker Standalone → Portainer Agent**.
3. Enter the server's display name and its address: `<REMOTE-SERVER-IP>:9001`.
4. Click **Connect**.

The remote host appears immediately in the environment list and can be steered with all functions.

Maintenance, backup and troubleshooting

Updating Portainer CE safely

Because Portainer runs in a container, updates happen by replacing the Docker image. All configuration data remains in the `portainer_data` volume:

```
# 1. Stop the existing Portainer container
sudo docker stop portainer

# 2. Remove the old container (the volume remains fully intact!)
sudo docker rm portainer

# 3. Pull the latest LTS image from the registry
sudo docker pull portainer/portainer-ce:lts

# 4. Restart the container with identical parameters
```

```
sudo docker run -d \
  -p 8000:8000 \
  -p 9443:9443 \
  --name portainer \
  --restart=always \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v portainer_data:/data \
  portainer/portainer-ce:lts
```

Full database backup

1. **Through the graphical interface:**

Navigate to **Settings → Backup Portainer**, choose *Download backup file* and optionally protect the archive with a password.

1. **Automated through the terminal (volume backup):**

```
# Backs up the contents of the portainer_data volume into a compressed tar archive
sudo docker run --rm \
  -v portainer_data:/data:ro \
  -v $(pwd):/backup \
  alpine tar -czf /backup/portainer_backup_$(date +%F).tar.gz -C /data .
```

Solving typical problems

□□ Practical example:

1. Security timeout on first start: If the message *“Your Portainer instance timed out for security purposes”* appears, the 5-minute window after container start was exceeded:

```
# Restart the container to reset the 5-minute counter
sudo docker restart portainer
```

2. Port collision on 9443: If another web service is already running on the server, it blocks port 9443:

```
# Check which process occupies the port
sudo ss -tulpn | grep 9443
```

Solution: map the external host port to a free port (e.g. `-p 10443:9443`) or put Portainer behind Nginx.

3. Cleaning unused disk space: Old image layers and stopped containers fill the system disk:

```
# Thorough cleanup of all unused images, build caches and stopped containers
sudo docker system prune -af --volumes
```

Command Reference (Cheatsheet)

Command / invocation	Area	Function and purpose in daily work

`docker volume create portainer_data`	Storage	Creates the persistent storage volume for the Portainer database
`docker run -d -p 9443:9443 ... portainer-ce:lts`	Setup	Starts the Portainer server in secured LTS mode
`docker run -d -p 9001:9001 ... agent:lts`	Multi-node	Starts the Portainer Agent on remote Docker hosts
`sudo docker ps --filter "name=portainer"`	Diagnosis	Checks the running and port status of the Portainer container
`sudo docker logs -f portainer`	Diagnosis	Streams live logs of the Portainer engine in the terminal
`sudo docker restart portainer`	Maintenance	Restarts Portainer (also resets the 5-minute setup timeout)
`sudo docker stop portainer && sudo docker rm portainer`	Update	Removes the old container instance before applying an update
`sudo docker pull portainer/portainer-ce:lts`	Update	Downloads the current LTS image before restart
`sudo docker volume inspect portainer_data`	Storage	Shows the actual storage path of the volume on the host
`docker system prune -af --volumes`	Hygiene	Cleans all unused images, containers and orphaned volumes
`sudo ss -tulpn	grep -E "(9443|8000)"`	Network	Verifies the active network sockets on the Ubuntu host
`sudo ufw status verbose`	Security	Checks the active UFW firewall rules for Portainer

Further Resources

Resource / documentation	Type	Description / purpose

Official Portainer documentation	Handbook	Complete reference manual for Portainer CE and Business Edition
Portainer GitHub repository	Source code	Official release notes, bug tracker and community discussion
Docker Compose specification	Reference	Official standard for modern compose.yaml multi-container stacks
Install Docker on Ubuntu	Practice guide	Step-by-step guide to a clean Docker Engine setup
Linux server hardening	Security	Best practices for securing SSH, sockets and firewalls
Ubuntu 24.04 LEMP stack and Nginx	Web server	Setting up Nginx as a performant reverse proxy with SSL/TLS

Conclusion

Portainer CE builds a reliable bridge between the uncompromising flexibility of the Linux command line and the wish for a central, clear control surface. On a server with Ubuntu 24.04 LTS the management platform can be set up in a few minutes and needs hardly any system resources in ongoing operation.

Through native support for modern **Docker Compose stacks**, web-based console access into containers and the ability to steer a fleet of remote servers through the **Portainer Agent**, Portainer becomes an indispensable tool for administrators and DevOps teams. Anyone who has understood the security implications of the Docker socket and runs Portainer behind a hardened Nginx reverse proxy gets a durable, high-performance platform.

> □□ **Practical tip for production:** For business-critical container environments always use the `:lts` tag instead of `:latest`. That avoids unforeseen breaking changes from monthly STS feature rollouts and gives you proven long-term stability.

If you want to harden your container infrastructure beyond graphical management, our guide to Linux server hardening with FIDO2 and CrowdSec shows how you shield the foundation of the host system against unauthorized access and automated attacks.

Source: https://admindocs.de/en/linux-beginners/install-portainer-on-ubuntu-24-04-lts