PORTAINER.iO

Whitepaper

RUNNING DOCKER SWARM IN 2026

The risks, the limitations, and your options

Mar 2026

FOREWORD

For many organizations, Docker Swarm was the first practical way to run containers in production. It provided simple orchestration, integrated cleanly with Docker, and allowed teams to move from virtual machines to containerized services without the operational overhead of a complex platform.

Thousands of environments still run Docker Swarm today. In many cases, those clusters quietly power critical internal systems, edge deployments, or legacy application platforms.

However, recent changes in the Docker ecosystem—particularly the release of Docker Engine v29—have introduced breaking changes and compatibility issues that disproportionately affect Swarm environments.

This whitepaper explains what has changed, the risks operators should understand, and the practical options available for organizations running Swarm in 2026.

WHY DOCKER SWARM IS BACK IN THE SPOTLIGHT

With the release of Docker Engine v29, several changes have affected compatibility across the Docker ecosystem. Many of these changes impact tooling, plugins, and workflows that Swarm users rely on for production environments.

In particular, the changes around API compatibility and storage plugins have created new operational risks for Swarm operators.

These issues do not mean that Swarm has stopped working. Many clusters continue to operate reliably. However, they do highlight how the broader ecosystem around Swarm is evolving, and why operators should review their long-term platform strategy.

BREAKING CHANGE: MINIMUM API VERSION RAISED

Docker Engine v29 increased the minimum supported daemon API version to v1.44.

This change removed compatibility with any client compiled against Docker Engine versions earlier than v25.

In practice, this means:

- Older Docker tooling can no longer communicate with the Docker daemon.
- Legacy plugins compiled against older APIs are rejected outright.
- There is no fallback compatibility layer.

Any tool, plugin, or platform built against an older API version receives a hard rejection from the daemon.

While this change affects the broader Docker ecosystem, it has an outsized impact on Swarm environments because they frequently rely on cluster-wide plugins and integrations that may not have been updated.

STORAGE PLUGIN FAILURES AND DATA ACCESS LOSS

One of the most serious issues operators have encountered involves legacy Docker volume plugins.

Many Swarm deployments rely on older **V1 volume plugins** to integrate with enterprise storage systems such as:

- NFS
- iSCSI
- SAN / NAS infrastructure

These plugins were often compiled against older Docker APIs and are now rejected by Docker Engine v29.

Because Docker Swarm requires **cluster-scoped persistent** storage, these plugins are a critical component of many production environments.

The failure mode can be alarming.

When the plugin fails to load:

- Docker still retains the volume metadata.
- However, the daemon cannot communicate with the storage plugin.
- As a result, volumes appear inaccessible.

Importantly, the underlying data is typically **not lost**. It remains intact on the storage backend but cannot be mounted until the plugin communication is restored.

Immediate mitigation

If you encounter this issue:

1. Do **not** delete `/var/lib/docker`.
2. Downgrade to Docker Engine v26 or v27.
3. Restore plugin communication.
4. Export or migrate the data.
5. Hold the Docker package version while planning a longer-term migration.

This issue has already caused major disruptions. In one case we are aware of, a very large enterprise experienced a significant site-wide outage after encountering this storage plugin failure.

ACTIVE BUGS AFFECTING SWARM CLUSTERS

Beyond compatibility issues, there are several known bugs affecting Swarm environments running Docker v29.

Mixed-version cluster networking failures

Clusters where some nodes run Docker v29 and others run older versions may encounter a networking failure.

In affected environments:

- encrypted overlay traffic silently stops between nodes
- inter-node communication breaks
- services may appear healthy but become unreachable

This effectively removes the ability to perform staged upgrades in clusters using encrypted overlay networks.

Operators must upgrade **all nodes simultaneously** to v29.1.2 or later to avoid this issue.

Standalone containers cannot communicate across nodes

A long-standing bug affects standalone containers attached to Swarm overlay networks.

Containers deployed outside Swarm services may hang when attempting to communicate across nodes.

The current workaround is to deploy cross-node workloads as **Swarm services rather than standalone containers**.

STRUCTURAL LIMITATIONS IN DOCKER SWARM

Beyond specific bugs and compatibility issues, Swarm has several architectural limitations that operators should consider when planning long-term infrastructure.

nftables incompatibility

Swarm currently cannot operate on hosts using **nftables**, which is now the default firewall framework on many modern Linux distributions.

To run Swarm, operators must enable legacy **iptables compatibility layers**, adding complexity to host configuration and future upgrades.

Overlay networking limits

Swarm relies on VXLAN-based overlay networking.

In large environments, overlay networks can become unstable once deployments exceed roughly **1,000 containers**.

These limits are largely tied to kernel-level VXLAN behavior and can introduce operational instability at scale.

VXLAN conflicts in modern infrastructure

VXLAN is widely used in modern infrastructure platforms, including:

- cloud networking systems
- virtualization platforms
- software-defined networks

Because Swarm also relies on VXLAN, port conflicts can occur if the default VXLAN port is not adjusted during configuration.

Operators deploying Swarm on infrastructure that already uses VXLAN should ensure the overlay networking port is changed to avoid collisions.

For example, both Swarm and VMware use UDP port 4789 for their VXLAN layer. To avoid the conflict, you can specify a different port number for Swarm to use when initiating the swarm:

```
docker swarm init --data-path-port=9789
```

WHAT THIS MEANS FOR SWARM OPERATORS

Taken together, these issues highlight three important realities about running Swarm today:

1. **SWARM ITSELF REMAINS FUNCTIONAL**.
   Many clusters continue to run stable production workloads.

2. **THE SURROUNDING ECOSYSTEM IS SHRINKING**.
   Fewer tools, plugins, and integrations are actively maintained.

3. **PLATFORM DECISIONS NOW REQUIRE MORE PLANNING**.
   Changes in Docker Engine and infrastructure tooling mean operators need to carefully manage upgrades and compatibility.

This does not necessarily require immediate migration, but it does mean organizations should begin evaluating their long-term container platform strategy.

YOUR OPTIONS MOVING FORWARD

Organizations currently running Docker Swarm generally have three practical paths forward. Each option has different operational trade-offs depending on the size of your platform, your upgrade cadence, and how actively you plan to evolve your infrastructure.

Option 1: Continue running Swarm with careful lifecycle management

Many environments will continue operating Swarm clusters successfully for years.

Some organizations will choose to continue running existing Swarm clusters, particularly for stable legacy workloads or edge deployments where the environment rarely changes.

If you take this approach, it becomes important to manage the platform more deliberately than in the past.

Practical steps

Operators running Swarm long-term should consider implementing the following controls:

Pin Docker Engine versions

Avoid automatic upgrades of Docker Engine packages. Maintain version pinning so that cluster upgrades can be tested before rollout.

Example approach:

- Hold Docker packages in your OS package manager
- Upgrade only after validating plugin compatibility in staging
- Avoid mixed-version clusters where possible

Audit volume plugins

Take an inventory of all Docker plugins currently used by your cluster.

Pay particular attention to:

- legacy **V1 volume plugins**
- plugins that have not been updated in several years
- storage integrations maintained by vendors that have moved on from Swarm

If possible:

- replace unsupported plugins
- migrate critical data volumes to standard storage backends
- ensure recovery procedures are documented

Document recovery procedures

Many Swarm environments were built several years ago and rely on institutional knowledge.

Operators should document:

- node recovery procedures
- cluster rejoin workflows
- volume recovery steps
- overlay network troubleshooting

These procedures become critical when ecosystem tooling is no longer actively maintained.

Evaluate firewall compatibility

If upgrading operating systems, confirm whether nftables will replace legacy iptables.

If so, ensure:

- iptables compatibility layers are enabled
- Swarm networking rules still function correctly

For organizations that plan to keep Swarm for several years, maintaining compatible host configurations becomes increasingly important.

Option 2: Introduce Kubernetes for new workloads

A common strategy is to keep existing Swarm workloads running while introducing Kubernetes for new applications.

This allows organizations to gradually adopt the Kubernetes ecosystem without disrupting stable services that already work.

Practical steps

Stand up a small Kubernetes environment

Start by deploying a small cluster for new workloads.

Many teams begin with:

- a single small production cluster
- a development cluster for testing deployments
- infrastructure automation for cluster creation

Lightweight Kubernetes distributions such as **k3s, Talos Linux**, or managed Kubernetes services can simplify initial deployment.

Deploy new services on Kubernetes

Rather than migrating existing services immediately, deploy new workloads on Kubernetes.

This approach allows teams to learn Kubernetes operational patterns while minimizing risk.

Typical early workloads include:

- new internal tools
- stateless web services
- CI/CD workloads
- experimental microservices

Adopt GitOps or declarative deployment

Kubernetes environments typically benefit from declarative deployment models such as GitOps.

This helps standardize:

- application deployment
- configuration management
- rollback procedures
- auditability of changes

Operate both platforms in parallel

During the transition period, teams may run:

- existing applications on Swarm
- new workloads on Kubernetes

Running both orchestrators is operationally manageable when they are controlled through a single management layer.

Platforms such as Portainer allow teams to operate Swarm and **Kubernetes side-by-side** while maintaining consistent access control and operational visibility.

Option 3: Plan a gradual migration away from Swarm

Organizations with large Swarm deployments or long-term infrastructure plans may prefer to migrate fully to Kubernetes.

Successful migrations rarely happen all at once. The safest approach is a phased transition.

Practical steps

Inventory current workloads

Begin by documenting the services currently deployed in your Swarm clusters:

- number of services
- stateful vs stateless workloads
- external dependencies
- storage requirements
- networking assumptions

This inventory helps determine migration complexity.

Identify low-risk services

Start migration with services that are:

- stateless
- internally facing
- loosely coupled

These workloads are usually easiest to re-deploy in Kubernetes.

Translate stack files

Most Swarm deployments are defined using **Docker Compose / stack files**.

These can often be translated into Kubernetes manifests or Helm charts.

During this stage:

- convert service definitions
- define persistent volumes
- configure service exposure (Ingress / LoadBalancer)
- replicate secrets and configuration management

Run both platforms during migration

For a period of time, organizations often run both Swarm and Kubernetes clusters.

This allows workloads to be moved gradually rather than forcing a high-risk migration window.

During this phase, teams can:

- move services incrementally
- validate monitoring and observability
- refine deployment pipelines

Retire Swarm clusters once workloads are migrated

Once all services have been migrated, Swarm clusters can be decommissioned in a controlled way.

This avoids rushed migrations and allows teams to evolve their operational practices alongside the platform transition.

WHY MANY TEAMS CHOOSE THE HYBRID PATH

In practice, many organizations follow a hybrid model:

- existing Swarm workloads remain operational
- new services are deployed on Kubernetes
- older services migrate over time

This approach avoids disruptive migrations while still aligning infrastructure with the broader container ecosystem.

Platforms such as **Portainer** make this transition significantly easier by providing a unified control plane capable of managing both Swarm and Kubernetes environments from a single interface.

This allows teams to modernize their infrastructure at their own pace without losing operational control of the systems already in production.

PORTAINER'S POSITION

Portainer fully supports both Docker and Docker Swarm.

Portainer has been fully compatible with Docker v29 since versions 2.33.5 LTS and 2.36.0 STS and users should ensure they are running these versions of Portainer or greater before upgrading Docker hosts.

However, given the combination of:

- ecosystem plugin failures
- active Swarm bugs
- structural networking limitations

operators should carefully evaluate their long-term platform strategy.

Portainer supports both Docker Swarm and Kubernetes from a single control plane.

This allows organizations to:

- operate existing Swarm clusters safely
- introduce Kubernetes gradually
- manage both environments through a single interface

For many teams, this hybrid model offers the safest path forward.

This enables teams to evolve their infrastructure without forcing disruptive migrations.

FINAL THOUGHTS

Docker Swarm solved an important problem for the industry: it made container orchestration simple and accessible.

Many production environments still rely on that simplicity today.

However, the container ecosystem has changed. Platform teams now need to manage not only container deployments, but also networking, storage integration, security, and long-term platform evolution.

Understanding the risks and limitations of running Swarm in 2026 is the first step toward making informed infrastructure decisions.

Whether you continue operating Swarm, introduce Kubernetes, or plan a gradual migration, the key is maintaining operational control over the environments you run today.

PORTAINER.iO