Trust model for Compose files

- The dependency chain
- Best practices
  - Inspect the full configuration
    - Fields to look out for
  - CI/CD environments
  - Pin remote references to digests
  - Other
- Further reading

Docker Compose treats every Compose file as trusted input. When a Compose file requests elevated privileges, host filesystem access, or any other configuration, Compose applies it as written. This is the same behavior as passing flags directly to `docker run`.

This means that any Compose file you run, whether it lives on your local filesystem, in a Git repository, or in an OCI registry, has full control over how containers interact with your host. The security boundary is not where the file comes from but whether you trust the author.

Evaluating trust means asking: Who authored this file? Has it changed since you last reviewed it? Do you understand every privilege it requests?

The dependency chain

A Compose application can be assembled from multiple sources. The `include` directive imports entire Compose files, while `extends` inherits configuration from a specific service in another file. Both support remote references and can be chained:

```
Your command
  └─ compose.yaml                                    (local or remote)
```

```
├─ services, volumes, networks                (direct config)
├─ include:
│    └─ oci://registry.example.com/base:v2   (remote dependency)
│         └─ services, volumes, networks      (indirect config)
└─ services:
     └─ app:
          └─ extends:
               └─ file: oci://registry.example.com/templates:v1
                    └─ service: webapp        (inherited config)
```

Each level has the same capabilities. The top-level file you inspect may appear safe while a nested `include` or `extends` introduces services with elevated privileges, host bind mounts, or untrusted images. These dependencies can also change independently. Risky settings can be introduced by a nested dependency that you never see unless you inspect the fully resolved output.

> Compose warns you when a configuration references remote sources. Do not accept this without understanding every reference in the chain.

Best practices

Inspect the full configuration

To see exactly what Compose applies, including all resolved `includes`, `extends`, merged overrides, and interpolated variables, use:

```
$ docker compose config
```

For remote references:

```
$ docker compose -f oci://registry.example.com/myapp:latest config
```

Review this output before running `up` or `create`, especially when the configuration comes from a source you have not audited.

Fields to look out for

A Compose configuration has broad control over how containers interact with the host. The following is a non-exhaustive list of fields that carry security implications when set by an untrusted

author:

Field	Effect
`privileged`	Grants the container full access to the host
`cap_add`	Adds Linux capabilities such as `SYS_ADMIN` or `NET_RAW`
`security_opt`	Configures security profiles including seccomp and AppArmor
`volumes` / bind mounts	Mounts host directories into the container
`network_mode: host`	Shares the host network stack
`pid: host`	Shares the host PID namespace
`devices`	Exposes host devices to the container
`image`	Pulls and runs an arbitrary container image
`env_file`, `label_file`, `secrets` / `configs` (`file:`), `include`, `extends`	Read files from the host, directly or through symlinks resolved from a remote checkout, and can surface their contents during configuration loading
`provider`	Runs the binary named by `provider.type` on the host, outside any container, when you run `up`, `down`, or `stop`

When in doubt, look up the effect of any unfamiliar field before running the configuration.

Like `volumes`, the file-reference fields read any file the user running Compose can access, including through symlinks resolved from a remote checkout, and their contents can appear in `docker compose config` output before any container starts. Compose does not confine reads to the project directory. Treat a Compose project as code you run, not data you inspect.

CI/CD environments

Automated pipelines are particularly sensitive because they often run with access to credentials, cloud provider tokens, or Docker sockets.

- Avoid referencing public or unverified Compose configurations in automated pipelines.
- Gate updates behind your normal code review process.
- Use read-only Docker socket mounts where possible to limit your risk.

Pin remote references to digests

Tags are mutable, meaning anyone with push access to a registry can overwrite a tag silently, so a reference you reviewed last week may point to different content today.

Digests are immutable. Instead of referencing by tag, pin to the digest.

```
include:
  - oci://registry.example.com/base@sha256:a1b2c3d4...
```

Treat any update to a pinned digest as a code change. Make sure you review the new content before updating the reference.

Other

- Use a private registry: Host OCI artifacts on a registry your organization controls. Restrict who can push to it.
- Audit transitive dependencies: Check every remote `include` and `extends` reference in the chain, not just the top-level file.
- Review all Compose confirmation prompts: When loading remote Compose files, Compose displays confirmation prompts for interpolation variables, environment values, and remote includes. Review these before accepting.

Further reading

- OCI artifact applications
- Use Compose in production
- `include` reference
- `extends` reference
- Manage secrets in Compose

□□□ #2
□ □□□ □□□ 2026-10-31 19:09:30 UTC
□ □□□ □□□ 2026-10-31 19:09:32 UTC