CHECKLIST

Docker Container Security Best Practices for Image Hardening Checklist

A practical, vendor-neutral checklist for reducing Docker image attack surface, removing unnecessary build and runtime components, and validating that a hardened image is fit for production deployment.

Docker Image Hardening Checklist

Use this checklist to harden Docker images before they reach production. Focus on reducing attack surface, removing unnecessary components, and validating that the final image behaves as expected at runtime.

1) Start with a trusted base image

- [ ] Choose the smallest base image that fully supports the application.
- [ ] Prefer maintained images with a clear update cadence and security support.
- [ ] Pin the base image by digest or otherwise control updates intentionally.
- [ ] Verify the base image contains only the runtime dependencies you need.
- [ ] Avoid using a general-purpose distribution unless your workload truly needs it.

2) Separate build-time tools from runtime artifacts

- [ ] Use multi-stage builds so compilers and build tools do not ship in the final image.
- [ ] Copy only the required application artifacts into the runtime stage.

- [ ] Remove source code, test fixtures, and build intermediates from the runtime image.
- [ ] Do not include package managers, shells, or debugging tools unless there is a documented production need.
- [ ] Confirm the final image does not contain build-only dependencies.

3) Minimize packages and libraries

- [ ] Install only the packages required to run the application.
- [ ] Remove package caches, indexes, and temporary files after installation.
- [ ] Delete unused libraries and utilities from the final image.
- [ ] Review transitive dependencies for unnecessary extras.
- [ ] Keep the final layer set as small as the application allows.

4) Run the container as a non-root user

- [ ] Create a dedicated non-root user for the application.
- [ ] Set the image to run with that user by default.
- [ ] Verify the application does not require root to start or operate.
- [ ] Check file ownership for logs, cache directories, sockets, and PID files.
- [ ] Confirm writable paths are explicit and limited to what the app needs.

5) Control filesystem permissions

- [ ] Ensure only required directories are writable.
- [ ] Avoid broad write access to the filesystem.
- [ ] Verify read permissions are sufficient for certificates, config files, and runtime assets.
- [ ] Confirm the container does not silently create files in unexpected paths.

- [ ] Check that permission changes do not force the process to run as root.

6) Prevent secret leakage

- [ ] Do not bake credentials, tokens, or private keys into the image.
- [ ] Keep secrets out of the build context.
- [ ] Avoid copying environment files meant only for local development.
- [ ] Remove shell history, temporary config files, and packaging leftovers.
- [ ] Review image layers to confirm secrets were never added.

7) Validate what is actually present in the image

- [ ] Inspect the final filesystem contents after the build.
- [ ] Confirm that only expected binaries, libraries, and config files remain.
- [ ] Check for unexpected shells, package managers, or admin utilities.
- [ ] Verify the image contents match the intended runtime footprint.
- [ ] Treat image scanning results as a signal, not proof of safety.

8) Verify runtime assumptions

- [ ] Start the container in a staging-like environment.
- [ ] Confirm the process starts under the intended non-root user.
- [ ] Validate health checks and startup behavior.
- [ ] Test logging, file writes, and any temporary storage paths.
- [ ] Confirm the application does not need hidden privileges to function.

9) Check operational fit before release

- [ ] Confirm the image size is reasonable for the workload.
- [ ] Verify the image is predictable to run and easy to reason about.
- [ ] Ensure the build process is repeatable.
- [ ] Document any exceptions, such as retained tools or writable directories.
- [ ] Decide whether the remaining risk is acceptable for this workload and operating model.

10) Final production readiness review

- [ ] Base image is trusted, minimal, and intentionally selected.
- [ ] Runtime image contains only required artifacts.
- [ ] Build tools and package managers are absent from the final image.
- [ ] Non-root execution is confirmed.
- [ ] Writable paths are explicit and limited.
- [ ] Secrets are absent from image layers and build outputs.
- [ ] Image contents have been inspected.
- [ ] Runtime behavior has been validated.
- [ ] Security scan findings have been reviewed and understood.
- [ ] The image is approved for deployment based on documented risk.

Quick self-check

If you answer "yes" to all of the following, your image is in a much better position for production use:

- [ ] Did we remove everything the application does not need?
- [ ] Did we verify the application still runs correctly after removal?
- [ ] Did we avoid shipping build-time tools into production?
- [ ] Did we run the container as a non-root user?
- [ ] Did we keep secrets out of the image entirely?

- [ ] Did we inspect the final filesystem contents, not just the scan report?

Notes

Use this checklist alongside runtime controls such as read-only filesystems, least privilege, and network restrictions. Image hardening reduces what ships; runtime controls reduce what can happen after startup. Both matter, and neither fully replaces the other.