CHECKLIST

Common .NET Mistakes and How to Avoid Them: Production Readiness Checklist

A practical, vendor-neutral checklist for reviewing .NET projects before production use, with checks for versioning, configuration, dependencies, environment assumptions, observability, and release readiness.

Common .NET Mistakes and How to Avoid Them

Production Readiness Checklist

Use this checklist to review a .NET service or tool before deployment. It is designed to catch operational mistakes that compile cleanly but fail in real environments.

How to use this checklist

- Review each item against the current codebase, build pipeline, and deployment environment.
- Mark each item as Yes, No, or Needs verification.
- If any item fails, fix it in a non-production environment first.
- Re-run validation before release.

---

1) Build, runtime, and version control

Target framework and SDK are explicit

- [ ] The project file specifies the intended target framework.
- [ ] The required SDK version is documented or pinned.
- [ ] Developers and CI use the same SDK baseline.
- [ ] Build agents are not relying on accidental machine defaults.

Dependency versions are controlled

- [ ] Package versions are explicit and reviewed.
- [ ] Central package management or a lockfile is used where appropriate.
- [ ] Dependency updates are intentional, not incidental.
- [ ] The restore process resolves the same versions in local and CI builds.

Build is reproducible

- [ ] A clean restore succeeds.
- [ ] A clean build succeeds without manual intervention.
- [ ] The same build succeeds in the pipeline environment.
- [ ] Any build failure can be reproduced on demand.

Validation checks

- [ ] Confirm the reported SDK version matches the intended baseline.
- [ ] Confirm the target framework matches the deployment plan.
- [ ] Confirm restore output does not drift unexpectedly between environments.

---

2) Configuration and secrets

Configuration is environment-aware

- [ ] Development, test, and production settings are separated.
- [ ] Environment-specific values are not hard-coded in source.
- [ ] Safe defaults are used when appropriate.
- [ ] Missing critical values fail closed rather than falling back unsafely.

Secrets are protected

- [ ] No secret values are committed to source control.
- [ ] Sensitive values are loaded from a secure store or protected mechanism.
- [ ] Secret handling follows organizational policy.
- [ ] Secret exposure has been checked and remediated if needed.

Startup behavior is verified

- [ ] The application starts with the expected settings in each environment.
- [ ] The effective configuration can be inspected in a controlled environment.
- [ ] A missing required setting causes a visible failure.
- [ ] External resource settings are validated before first use.

Validation checks

- [ ] Test startup with expected configuration.
- [ ] Test startup with a required setting missing.
- [ ] Confirm no hard-coded credentials remain.
- [ ] Confirm production settings do not depend on developer-only values.

---

3) Dependency hygiene

Package references are intentional

- [ ] Each direct package reference has a documented purpose.
- [ ] Unused packages have been removed.
- [ ] Test-only and build-only dependencies are separated from runtime dependencies.
- [ ] Transitive dependencies are understood, not ignored.

Supply chain review is in place

- [ ] Package provenance or licensing checks are part of the update process if required.
- [ ] Vulnerability review is performed before release.
- [ ] Dependency updates are reviewed for compatibility impact.
- [ ] Hidden dependency upgrades are identified early.

Validation checks

- [ ] Restore the project and review the final dependency list.
- [ ] Remove one unused package and confirm the app still works.
- [ ] Confirm the pipeline policy or vulnerability gate still passes.
- [ ] Verify no package remains solely out of habit.

---

4) Environment assumptions

File paths and OS behavior are portable where required

- [ ] No hard-coded drive letters or machine-specific paths remain.
- [ ] File access uses configuration or platform-neutral APIs where possible.
- [ ] Local user profile assumptions are avoided.
- [ ] OS-specific requirements are documented if they are intentional.

Runtime location is not assumed

- [ ] The app is tested in the same platform type it will run in.
- [ ] Container execution is validated if the app is containerized.
- [ ] Linux, Windows, and cloud differences are considered where relevant.
- [ ] Certificate and network access work in the target environment.

Validation checks

- [ ] Run the app in the target runtime, not only on a developer workstation.
- [ ] Confirm startup, file access, and network calls work as expected.
- [ ] Confirm shutdown behavior is correct in container or service mode.

- [ ] Confirm no environment-specific shortcut is hiding a portability problem.

---

5) Observability and failure signals

Errors are visible and actionable

- [ ] Startup failures are clearly reported.
- [ ] Exceptions are captured with useful context.
- [ ] Logs provide enough detail to diagnose failures.
- [ ] Correlation is available for requests, jobs, or background tasks where needed.

Logs are safe

- [ ] Logs do not expose secrets or tokens.
- [ ] Logs do not expose personally identifiable data unless policy allows it.
- [ ] Logging level is appropriate for production use.
- [ ] Error messages are informative without being overly verbose or sensitive.

Metrics and signals exist where needed

- [ ] Operational metrics are available for critical behaviors.
- [ ] Failure conditions can be distinguished from transient issues.
- [ ] Alerting or monitoring can detect startup or runtime failures.
- [ ] Missing telemetry would not leave the system opaque during an incident.

Validation checks

- [ ] Trigger a controlled failure and confirm it is visible.
- [ ] Verify logs include enough context to identify what failed and where.
- [ ] Verify no sensitive data appears in logs.
- [ ] Confirm operational staff can tell whether a failure is transient or structural.

---

6) Release and deployment readiness

Pre-production checks are complete

- [ ] The latest changes have been tested in a non-production path.
- [ ] Rollback or recovery steps are documented.
- [ ] Deployment dependencies are understood.
- [ ] The release is not blocked by unresolved warnings or unknown changes.

Change impact is understood

- [ ] Configuration changes have been reviewed separately from code changes.
- [ ] Dependency updates have been tested for behavior changes.
- [ ] Build and runtime changes have been validated in the same environment class as production.
- [ ] The team knows what could fail after deployment.

Final readiness check

- [ ] Build is reproducible.
- [ ] Configuration is explicit.
- [ ] Secrets are protected.
- [ ] Dependencies are intentional.
- [ ] Environment assumptions are tested.
- [ ] Observability is sufficient.
- [ ] Rollback path exists.

Quick review summary

Use this summary before approving release:

- Versioning: SDK, framework, and package versions are explicit and consistent.
- Configuration: Environment values are separated and secrets are controlled.
- Dependencies: Only necessary packages remain, and transitive risk is understood.
- Portability: No hidden machine-specific assumptions remain.
- Observability: Failures are visible, diagnosable, and safe to log.
- Readiness: The app has been verified in a non-production environment.

Recommended action if any item fails

- Fix the issue in a controlled environment.
- Rebuild and retest from a clean state.
- Validate both success and failure paths.
- Confirm the pipeline behaves the same way as local testing.
- Do not deploy until the issue is understood and reproducible behavior is restored.

Notes

- Treat unexpected behavior as a deployment risk, not just a code issue.
- Prefer explicit settings over implicit defaults.
- Validate the application where it will actually run.
- Keep the checklist as part of your release process for repeatable reviews.