HOW-TO GUIDE

How to get started with .NET

A practical, step-by-step guide to installing .NET, creating a first application, validating the runtime, and confirming a safe setup before moving to production workflows.

Programming - July 02, 2026

Quick start

If you need to get started with .NET quickly, the operational goal is simple: install the SDK, confirm the tooling works, create a project, run it locally, and verify that your environment matches the version you expect to use in development or automation.

In practice, that means you should be able to complete four checks before you spend time writing application code:

- dotnet --info shows the expected SDK and runtime versions.
- A new project can be created from a template.
- The project builds without errors.
- The app runs locally and returns the expected output.

This guide focuses on that workflow. It shows you how to set up a working .NET environment, validate it, and avoid common operational mistakes such as using the runtime instead of the SDK or mixing incompatible versions across machines.

What you need before you begin

Before installing anything, confirm what you actually need to do with .NET. The setup is slightly different depending on whether you only want to run an app or you plan to build and modify one.

- If you will compile code or create new projects, install the SDK.
- If you only need to host an already-built app on a server, you may only need the runtime.
- If you are using CI/CD, automation, or security scanning, pin a specific version and document it.

For most developers and engineers, the SDK is the correct starting point. It includes the compiler, project templates, and the dotnet CLI. If your environment must be standardized, write down the version requirement before installation so that workstations, build agents, and containers all align.

If you want a concise operational explanation of what .NET is used for and how the runtime, libraries, and tooling fit together, What is Programming in .NET? A Practical Operational View is a useful companion.

Install the SDK

Install the latest supported SDK version for your operating system from the official distribution channel you already use in your environment. For managed environments, that may mean a package manager, endpoint management tool, container image, or manual installer.

The important decision is not the installer itself; it is whether the version you install matches the version you will support in development and deployment.

Verify the installation

After installation, open a new terminal and run:

You should see:

- the installed SDK version
- the installed runtimes
- the operating system information

- architecture details such as x64 or arm64

For a cleaner version check, you can also use:

Expected output is a single version number such as 8.x.x or another version you installed.
If the command is missing, the SDK is not on your PATH, or the installation did not complete successfully.

Operational boundary

If you are preparing a production image, CI runner, or locked-down workstation, do not assume the latest SDK is the right one. Verify:

- the exact version supported by the application
- whether your build pipeline needs one SDK or multiple side-by-side versions
- whether your security baseline allows the installer source and package type

For deployment readiness questions, use a checklist approach rather than ad hoc validation. A structured review such as NET Checklist: Operational Readiness Review for Production Use helps confirm build, security, observability, and rollback expectations before release.

Create your first project

Once the SDK is installed, create a new project from a template. This is the fastest way to confirm the toolchain is functional end to end.

A common example is a console application:

What this does:

- creates a new project directory named HelloDotNet
- generates a project file and starter source file
- prepares the project for build and execution with the CLI

If the template command fails, the most likely causes are:

- the SDK is not installed
- the CLI is not on the PATH

- your environment has restricted access to template packs or first-run setup

Inspect the generated files

A basic console project usually includes:

- a project file such as HelloDotNet.csproj
- a source file such as Program.cs

You do not need to memorize every file yet. The key point is that .NET project structure is explicit and reproducible. That matters operationally because build systems, code review, and dependency management all rely on that project file.

Build the project

Build the project to confirm the compiler and dependency restore process work correctly.

Expected output is a successful build with no errors. You may see warnings, depending on template defaults and target framework settings, but warnings should be reviewed rather than ignored.

If the build fails, check the following first:

- the SDK version is installed as expected
- NuGet/package restore is allowed by network policy
- your machine can access any required package feeds
- the target framework in the project file matches the installed SDK support

For a new setup, do not move on until the build succeeds locally. A working build is the most reliable proof that the SDK, compiler, restore path, and project configuration are all aligned.

Run the app locally

Use the CLI to start the app:

For the default console template, the expected result is simple output in your terminal. If you changed the starter code, confirm that the app prints or returns what you intended.

A practical validation pattern is:

- run it once after creation
- edit the code
- run it again
- confirm the output changes as expected

That sequence proves more than a successful build. It confirms the project is editable, the runtime launches correctly, and your local development path is functioning.

Make a small change and validate it

After the first successful run, change the starter program to prove that your environment supports a normal edit-build-run cycle.

For example, replace the default output with something recognizable:

Then run:

Expected output:

If the output does not change, validate the following:

- the correct project directory is open
- the source file you edited is the one used by the project
- your editor saved the file
- the build succeeded before execution

This is a small test, but it is operationally important. It confirms that source control, editor behavior, build tooling, and runtime execution are all in the same path.

Understand the minimum commands you will use often

You do not need the full .NET command surface to get started. These commands cover the essential local workflow:

Check installed SDKs and runtimes.

Create a new console project.

Compile the project and restore dependencies.

Build if needed and run the app.

Run tests when your project includes them.

Use these commands as your baseline validation set. If one of them fails, stop and fix the environment before introducing application complexity.

Decide whether your environment is production-ready

A local setup that works interactively is not automatically safe for production use. Before promoting a project, confirm a few practical controls.

Version control

Pin the SDK version used by the team or pipeline. Verify that all contributors and build agents use the same supported version unless you intentionally allow multiple versions.

Dependency provenance

Check where packages are restored from and whether that source is approved. In restricted environments, a blocked package feed often looks like a build failure even though the code is valid.

Build reproducibility

Run the same build command on a clean environment if possible. The goal is to ensure the project does not rely on hidden local state.

Security and rollback

If the app will be deployed, confirm how you will roll back to the previous version if the new build misbehaves. Keep the previous package or artifact until the new release is verified in production-like conditions.

A disciplined operational approach matters here more than code style. The setup is complete only when you can explain which version is installed, how the build is produced, and how you would revert it safely.

Common issues and how to check them

Most first-time .NET setup problems are straightforward once you know what to inspect.

dotnet command not found

This usually means one of three things:

- the SDK is not installed
- the command is not in the shell PATH
- the terminal session has not been restarted after installation

Reopen the terminal and run dotnet --version again.

Build fails after project creation

Check the project target framework and SDK version. A mismatch between the project configuration and installed toolchain is a common cause of failures.

Restore fails

If package restore fails, verify network access, proxy settings, authentication to private feeds, and certificate trust if your environment intercepts outbound traffic.

App runs but output is unexpected

Confirm that you edited the correct file and that the code change was saved before running again.

In security-restricted or containerized environments, also verify that filesystem permissions allow the build to create output under the working directory.

Cleanup and safe rollback

If you were only validating the setup, cleanup is simple.

- Delete the sample project directory if it is no longer needed.
- Remove any temporary package feeds or credentials you used for restore testing.
- Uninstall the SDK only if you installed it for evaluation and do not need it for ongoing work.

If you installed multiple SDK versions, keep the one required by your active project and document any older versions that remain for compatibility. Avoid removing a version that other projects or build agents still depend on.

Final takeaway

To get started with .NET safely and efficiently, install the SDK, confirm the CLI works, create a simple project, build it, run it, and verify the exact version you will support. That sequence gives you a reliable baseline before you move into application development, CI automation, or production deployment.

Once those checks pass, you have a working .NET environment you can trust for repeatable local development and controlled operational use.