Introduction to .NET

Session #01

dotnet training

Agenda

Intro to .NET

Introduction to .NET: components, versions

Getting started to develop in .NET

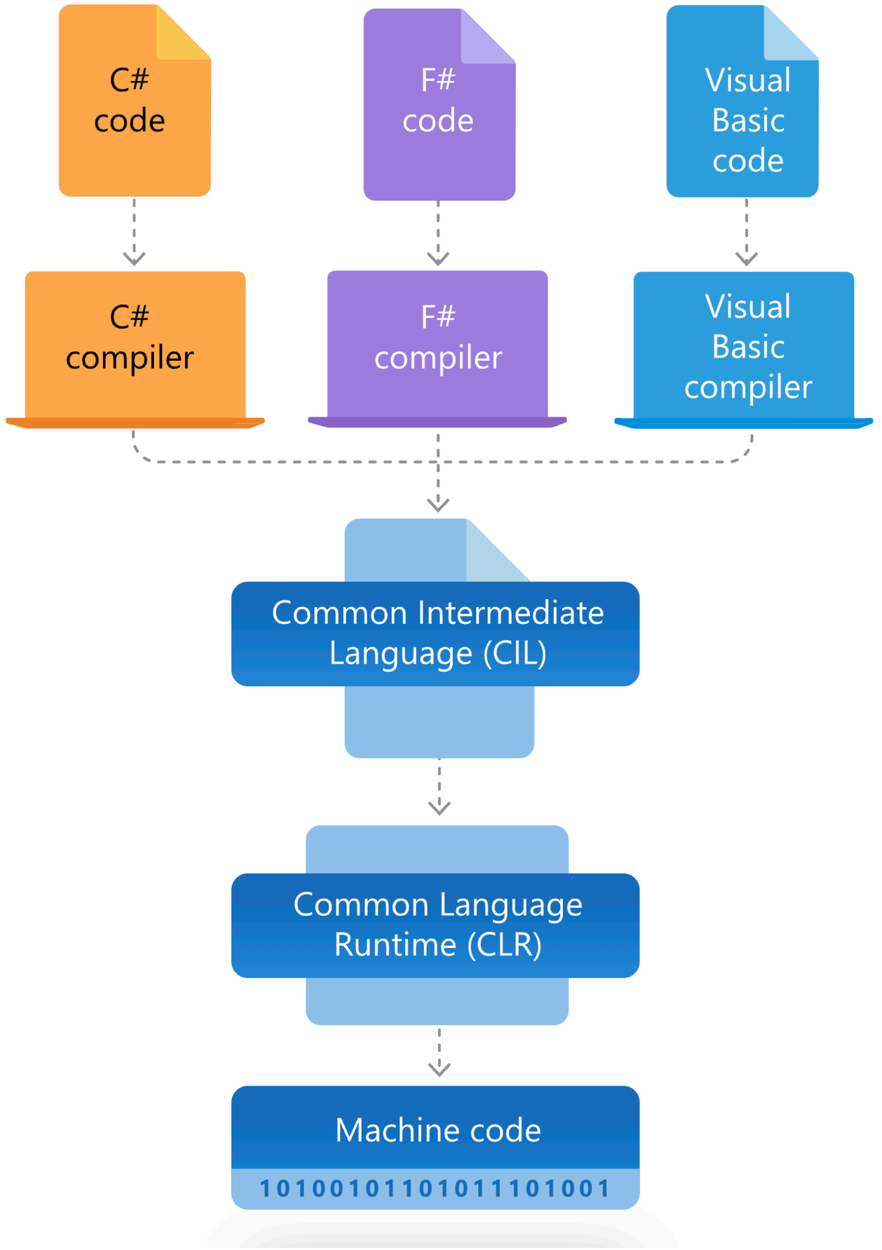
dotnet CLI

Nuget package management

Introduction to .NET

What is .NET?

- .NET is the free, open-source, cross-platform framework for ubuilding modern apps and powerful cloud services
- Have 3 programming languages: C#, Visual Basic, F#
- Code translate to IL (Intermediate Language) and runs on a CLR (Common Language Runtime)

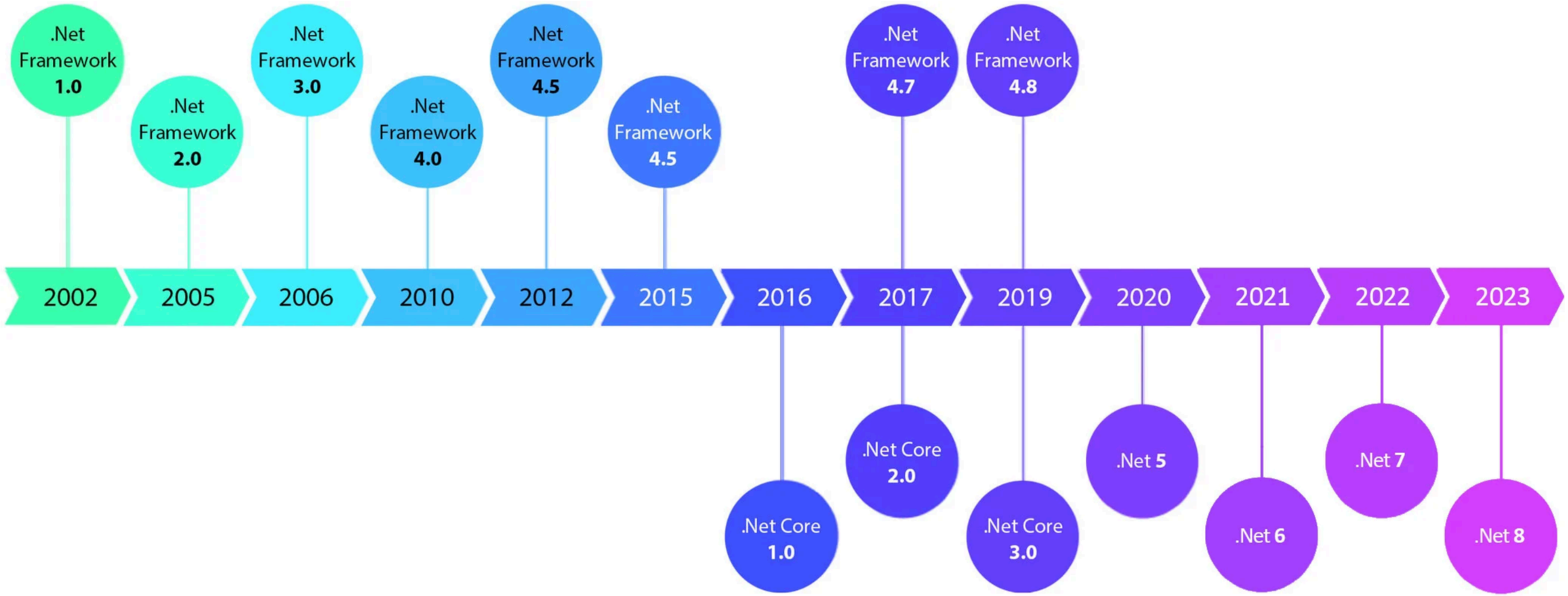


dotnet Versions

- .NET Framework
  - Initial framework and Windows-only version
  - Only minor/security fixes
  - 4.8 version will be supported when installed on a supported Windows versions
  - Needed for several Windows components (PowerShell, …)
- dotnet core
  - Initial cross-platform version
  - Available on Windows, Linux, MacOS
  - Existed in parallel with .NET Framework
- .NET:
  - Actual .NET version and the only to be continued
  - Cross-platform
  - Every year a new version with even numbers with LTS

.NET Timeline

Timeline

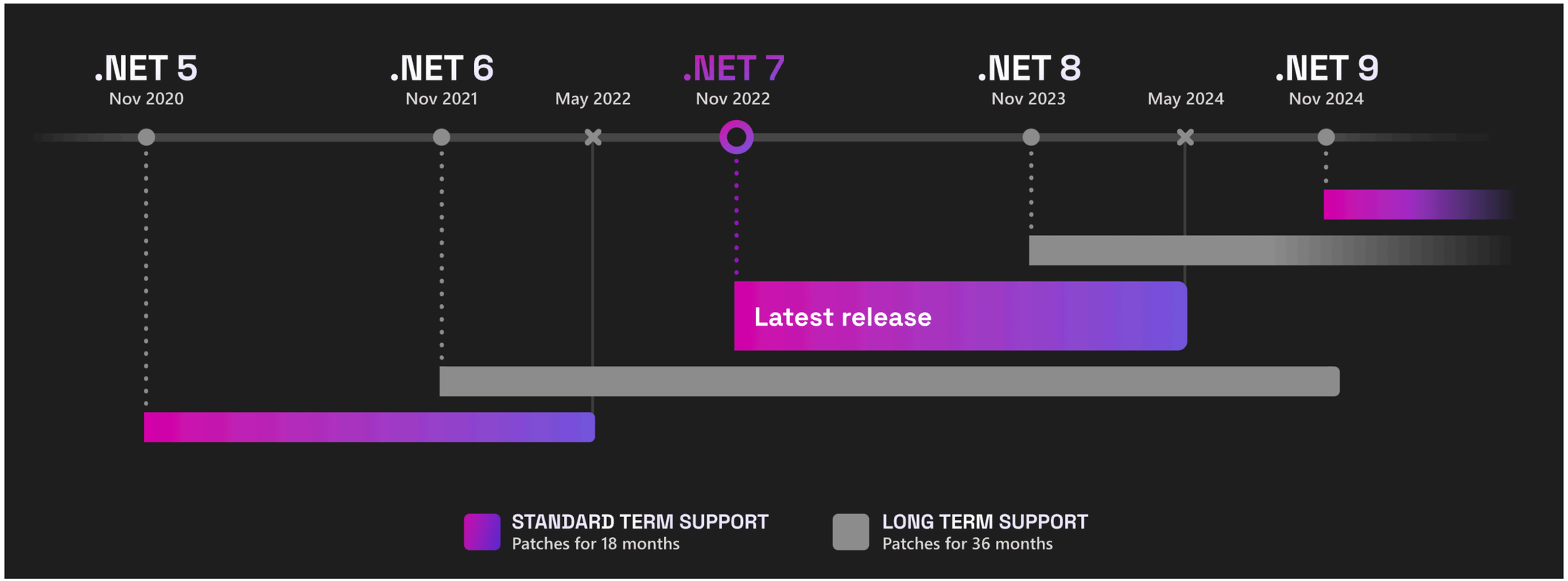


.NET Framework Lifecycle

Version	Start Date	End Date
.NET Framework 4.8.1	Aug 9, 2022	
.NET Framework 4.8	Apr 18, 2019	
.NET Framework 4.7.2	Apr 30, 2018	
.NET Framework 4.7.1	Oct 17, 2017	
.NET Framework 4.7	Apr 11, 2017	
.NET Framework 4.6.2	Aug 2, 2016	Jan 12, 2027

.NET Support

Timeline



.NET Vocabulary

Main components

- .NET SDK: Compiler and Runtime (like JDK)
- .NET runtime: Runtime (JRE)
- Roslyn: C# (and VB) compiler. Open-source available here
- ASP.NET: Web libraries for frontend and backend
- ASP.NET Web API: REST API components and specification
- Razor: programming syntax used to create dynamic web pages
- Blazor: a free and open-source web framework that enables developers to create web apps using C# and HTML (C# on the browser using WebAssembly)
- MAUI: .NET Multi-platform App UI allow you to use the same code to generate native UIs for Desktop, Android, iOS, etc (previous named Xamarim)

C sharp (C#)
Programming Language

- Object-oriented Programming Language
- Similar with JAVA
- Most used language on dotnet universe
- Usually a new version is released on new dotnet version
- Actually on version 11 (https://learn.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-version-history)

Getting Started

How to start?
SDK

- For SDK, recommended one is .NET 7.0
- Even 7.0 is STS, 8.0 will be released in November and will be LTS having all 7.0 features

OS	Installers	Binaries
Linux	Package manager instructions	Arm32 | Arm32 Alpine | Arm64 | Arm64 Alpine | x64 | x64 Alpine
macOS	Arm64 | x64	Arm64 | x64
Windows	Arm64 | x64 | x86 | winget instructions	Arm64 | x64 | x86
All	dotnet-install scripts	

How to start?
IDE

- Two main flavours, Visual Studio or VS Code
- VS Code is the most used Code Editor in the world and is fully integrated with .NET (and everything else…)
- Visual Studio is Microsoft IDE for .NET since the beginning
- VS Code is free and open-source
- Visual Studio have a free edition (Community) but don’t have all feature as pais versions
- When using Windows as developer machine, selecting one (if you have licences for Visual Studio) can be a case of particular choice
- On other operating systems, VS Code is the “only” choice

How to start?

IDE for JAVA Developers

- If you’re a JAVA developer using IntelliJ IDEA you can use Rider
- Rider is commercial IDE from JetBrains with UI/UX similar with IntelliJ
- JetBrains have a great library called ReSharper to help coding in .NET with auto-complete, snippets and other benefits
- On VS Code you have OmniSharper that does a similar (not so complete) work as ReSharper

How to start?

For the labs

- On this training you'll use VS Code + .NET 7.0 (first lab is to have everything running properly)
- Additional recommended extensions to use on VS Code
  - C# for Visual Studio Code (powered by OmniSharp)
  - C# Extensions
  - REST Client
  - GitHub Copilot

dotnet CLI

dotnet CLI

What is?

- .NET Command-line interface (CLI) is a cross-platform toolchain for developing, building, running, and publishing .NET applications
- Automatically installed with SDK
- When using Visual Studio you have an UI that makes this commands to you

dotnet new

.NET CLI

```
dotnet new <TEMPLATE>
```

- Creates a new project, configuration file, or solution based on the specified template
- The command calls the template engine to create the artifacts on disk based on the specified template and options
- Several templates available natively but additional ones can be used from community and even create your owns
- Templates: web, console, classlib, xunit, gitignore, …

dotnet build

.NET CLI

```
dotnet build
```

- Builds a project and all of its dependencies into a set of binaries
- The binaries include the project's code in Intermediate Language (IL) files with a *.dll* extension
- Depending on the project type and settings, other files may be included like an executable

dotnet build

.NET Configurations

- Build command uses two concepts: Configuration and Platform
- Platform indicates for which platform you want to build your code, like x86, ARM, etc. Default is ‘`Any CPU`’
- Configuration defines how the compiler should behave in specific configurations, like enabling `DEBUG` and `TRACE` flags
- By default, exists 2 configurations: `Debug` and `Release`
- `Debug` is the default but you should specify `Release` configuration when you are delivering your app to production

dotnet restore

.NET CLI

```
dotnet restore
```

- Restores the external dependencies and tools of a project
- External dependencies and versions are referenced in project file and used by Nuget tool to get them from package management service
- In most cases, you don't need to explicitly use the command, since a NuGet restore is run implicitly if necessary when you run the other commands like `dotnet build`, `dotnet run`, etc.

dotnet run
.NET CLI

```
dotnet run
```

- Provides a convenient option to run your application from the source code with one command
- If needed, a build command is automatically run (that if needed runs a restore command)
- Uses the concept of “build configurations” like build command using `Debug` as default
- Used only in dev/non-prod environments

dotnet publish

.NET CLI

```
dotnet publish
```

- Publishes the application and its dependencies to a folder for deployment to a hosting system
- Compiles the application, reads through its dependencies specified in the project file, and publishes the resulting set of files to a directory
- Uses the concept of “build configurations” but since `Debug` is the default, usually you set the build configuration parameter to `Release`
- The outcome is what you should use in production environments

dotnet test

.NET CLI

```
dotnet test
```

- CLI test driver used to execute unit tests
- Builds the solution and runs a test host application for each test project in the solution
- The test host executes tests in the given project using a test framework, for example: MSTest, NUnit, or xUnit, and reports the success or failure of each test
- f all tests are successful, the test runner returns 0 as an exit code; otherwise if any test fails, it returns 1

dotnet VS MSBuild

How those integrate?

- MSBuild is the real compiler app that makes possible build, run, test, etc. commands to be executed
- MSBuild have a hard interface to handle parameters and is an old application that stills supports commands from several years ago
- .NET CLI is a cleaner solution that serves as a proxy to MSBuild commands

dotnet add package

.NET CLI

```
dotnet add package
```

- Provides a convenient option to add or update a package reference in a project file
- When you run the command, there's a compatibility check to ensure the package is compatible with the frameworks in the project.
- If the check passes and the package isn't referenced in the project file, an element is added to the project file. If the check passes and the package is already referenced in the project file, the element is updated to the latest compatible version
- You can specify a version or automatically will use the newest version

dotnet add reference

.NET CLI

```
dotnet add reference
```

- Provides a convenient option to add project references to a project
- This command allow you to define dependencies between your projects (we can discuss if this is the best way to work on… :)

.NET Projects

.NET CLI

- When you create an app or website in Visual Studio, you start with a ***project***
- In a logical sense, a project contains all files that are compiled into an executable, library, or website
- Those files can include source code, icons, images, data files, and more.
- A project also contains compiler settings and other configuration files that might be needed by various services or components that your program communicates with, like external dependencies or project references
- Project file is a `.csproj` extension file and have a XML format following MSBuild XML Schema

.NET Solutions

.NET CLI

- A project is (can be) contained within a ***solution***.
- A solution it's simply a container for one or more related projects, along with build information, Visual Studio window settings, and any miscellaneous files that aren't associated with a particular project
- Solution files can be used to define properties and configuration common for several projects and allow us to easily run commands like build, run, test since they will be targeted to all projects within the solution
- ***Projects*** are mandatories. ***Solutions*** are optional but helpful

.NET Solutions

.NET CLI

- A solution is described by a text file (extension *.sln*) with its own unique format; it's not intended to be edited by hand
- Conversely, the *.suo* file is a hidden file that is not displayed under the default File Explorer settings

Extension	Name	Description
.sln	Visual Studio Solution	Organizes projects, project items, and solution items in the solution.
.suo	Solution User Options	Stores user-level settings and customizations, such as breakpoints.

dotnet sln

.NET CLI

```
dotnet sln <COMMAND>
```

- Lists or modifies the projects in a .NET solution file
- This is the recommended way to interact with solution files
- Available commands: `list`, `add`, `remove`

Nuget

Nuget

Package Management

- Nuget is the .NET platform package management
- Equivalent of Maven or Gradle in JAVA
- nuget.org is the public and official nugget package management website where you can find and get .NET packages
- Several other services allow you to have public/private repositories, like Azure DevOps, GitHub, Artifactory, Nexus, etc.

Where to define packages?

Nuget

- Packages are defined at project level
- For each package, you get a `<PackageReference>` entry on your .csproj XML file with version to be used
- By default, Nuget will use nuget.org to find the packages (configured at machine level when .NET SDK/Runtime is installed)
- If you want to use other repositories, you can create a `nuget.config` file where you specify which repository you want to use based on package name

How to interact with Nuget?

Nuget CLI

- Nuget have a Nuget CLI that allow you to interact with your project and the repositories
- With this CLI you can create new packages, push/pull to/from an external repository and to control your sources
- Like with MSBuild, .NET CLI allow you to use the same CLI to interact with Nuget executable

dotnet pack

.NET CLI

```
dotnet pack
```

- Packs the code into a NuGet package.
- The result of this command is a NuGet package (that is, a *.nupkg* file)
- NuGet dependencies of the packed project are added to the *.nuspec* file, so they're properly resolved when the package is installed
- If the packed project has references to other projects, the other projects are not included in the package. Currently, you must have a package per project if you have project-to-project dependencies.

dotnet nuget add source

.NET CLI

```
dotnet nuget add source
```

- Adds a new package source to your NuGet configuration files

dotnet nuget push

.NET CLI

`dotnet nuget push`

- Pushes a package to the server and publishes it
- The server to be used is defined on `nuget.config` file using the usual chain: machine (on .NET folder) and any `nuget.config` file from root of drive until actual directory
- An explicit `pull` command is not needed since is the expected behaviour of `restore` command

Q&A

Prepare your machine

Lab #01

dotnet training

Prepare your machine

Lab #01

- Learning Objectives
  - Install .NET SDK
  - Install VS Code
  - Add recommended extensions to VS Code
  - Run your first CLI command
  - Test your first .NET code
- MD Link: https://github.com/tasb/dotnet-training/blob/main/labs/lab01.md
- HTML Link: https://tasb.github.io/dotnet-training/labs/lab01.html