

# ÉVOLUTION DE L'ÉCOSYSTÈME .NET

DE .NET FRAMEWORK 4.8 À .NET 8/10

C# • ASP.NET CORE • TRAJECTOIRE LTS

# OBJECTIFS DE LA SESSION

- Comparer .NET Framework 4.8 vs .NET 8 côté Web API
- Voir les vraies avancées C# utiles au quotidien
- Identifier ce que ça change pour l'équipe maintenant

# AGENDA

- Plateforme : ce qui change vraiment de 4.8 à 8
- C# : évolutions majeures (7.3 → 12)
- Web API : delta concret (hosting, config, perf)
- Synthèse
- Questions

# PLATEFORME .NET : 4.8 VS 8

# .NET FRAMEWORK 4.8 AUJOURD'HUI

- Stable, supporté, fiable
- **Figé** : pas de nouvelles fonctionnalités
- Windows-only, IIS couplé

# .NET 8 (LTS)

- Cross-plateforme (Windows/Linux/containers)
- Kestrel par défaut, IIS ou Nginx en reverse proxy
- Config JSON + options, DI native, middleware pipeline
- Gains perf noyau (JIT, GC, sockets, HTTP/2/3)

# CE QUI CHANGE POUR LES DEVS

- SDK unifié : `dotnet build/test/publish`
- Projets SDK-style (`csproj` léger, références implicites)
- Tooling Roslyn/analyzers, nullable et warnings utiles
- Templates Web API modernes (controllers + minimal APIs)

# OUTILLAGE QUI AIDE (IMMÉDIAT)

- `dotnet format` + analyzers (warnings en errors sur CI)
- Hot reload pour les démos/itérations rapides
- Templates SDK-style et `dotnet new api`



# SÉCU / SECRETS (DEV)

- `dotnet user-secrets` pour le local
- Config par environnement  
(`appsettings.Development.json`)
- Certificats dev : `dotnet dev-certs https --trust`
- Pas de secrets dans `appsettings.json`

# C# 7.3 → 12 : LES GROS APPORTS

# PATTERN MATCHING

```
return status switch
{
    200 => "OK",
    404 => "Not Found",
    _   => "Error"
};
```

# RECORDS & WITH-EXPRESSIONS

```
public record UserDto(int Id, string Name);  
var updated = user with { Name = "Jane" };
```

# NULLABLE REFERENCE TYPES

```
string? name = GetName();  
if (name is not null) Console.WriteLine(name.Length);
```

# ASYNC/AWAIT PARTOUT + ASYNC STREAMS

```
var users = await _repository.GetAllAsync();  
return Results.Ok(users);
```

# SPAN/MEMORY (PERF SANS ALLOCATIONS)

```
ReadOnlySpan<byte> buffer = stackalloc byte[64];  
// Traitement sans allocations intermédiaires
```

# FAQ C#

- Obligatoire ? Non, opt-in par projet.
- Progressif ? Oui, feature par feature.
- Compatibilité ? Forte avec le code existant, compiler warn d'abord.



# WEB API : 4.8 VS 8

# ARCHITECTURE : HÉBERGEMENT / CONFIG

Web API 2 (.NET  
4.8)

ASP.NET Core

---

System.Web + IIS

Kestrel + middleware, IIS/Nginx  
en reverse proxy

---

Global.asax

Program.cs minimal

---

Web.config

appsettings.json + options

---

Hébergement  
Windows

Cross-plateforme, conteneurs

# ARCHITECTURE : PIPELINE / SÉRIALISATION

**Web API 2 (.NET 4.8)**

**ASP.NET Core**

---

HttpModules/Handlers

Middleware pipeline

---

DI externe

DI native (AddScoped,  
etc.)

---

# CONTROLLER CLASSIQUE EN ASP.NET CORE

```
[ApiController]
[Route("api/users")]
public class UsersController : ControllerBase
{
    private readonly IUserService _service;
    public UsersController(IUserService service) => _service = service;

    [HttpGet("{id}")]
    public IActionResult Get(int id) => Ok(_service.Get(id));
}
```

# MINIMAL API (POUR EXPOSER VITE)

```
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddScoped<IUserService, UserService>();

var app = builder.Build();
app.MapGet("/api/users/{id}", async (int id, IUserService svc) =>
{
    if (id <= 0) return Results.BadRequest();
    var user = await svc.GetAsync(id);
    return user is null ? Results.NotFound() : Results.Ok(user);
});

app.Run();
```

# PERF WEB API (.NET 8)

- System.Text.Json par défaut, source generators disponibles
- HTTP/2 et HTTP/3 supportés
- Output caching et rate limiting intégrés
- Kestrel + pooling sockets / GC server

# SOURCE-GEN JSON (EXEMPLE)

```
[JsonSerializable(typeof(UserDto))]  
internal partial class AppJsonContext : JsonSerializerContext;  
  
var json = JsonSerializer.Serialize(user, AppJsonContext.Default.
```

# PIPELINE MIDDLEWARE

- Logging structuré
- AuthN/AuthZ (UseAuthentication, UseAuthorization)
- CORS, compression, rate limiting
- Endpoints (controllers ou minimal APIs)



# FAQ WEB API

- Réécriture complète ? Non, métier conservé.
- Performances ? + grâce à Kestrel, pooling, JSON plus rapide.
- Hébergement ? IIS, Linux + Nginx, ou containers.

# CE QU'IL FAUT RETENIR

- .NET 8 apporte hosting moderne, perf et cross-plateforme — sans changer votre métier Web API
- C# moderne réduit le bruit (pattern matching, records, nullable) et sécurise la base
- Pipeline middleware + DI native simplifient l'infra et améliorent les perfs

# QUESTIONS



Scannez pour accéder à la présentation

© 2026 — Support pédagogique.

Usage formation et sensibilisation. Réutilisation ou diffusion externe à valider.

# RESSOURCES



## DÉPÔT GITHUB

[github.com/Nicolas-Cousin-Tech-Solutions/dotnet-modernization-overview](https://github.com/Nicolas-Cousin-Tech-Solutions/dotnet-modernization-overview)



## TÉLÉCHARGER LE PDF

[dotnet-modernization-overview.pdf](#)

© 2026 — Support pédagogique.

Usage formation et sensibilisation. Réutilisation ou diffusion externe à valider.