.NET Core 2.0

What's new and why it matters

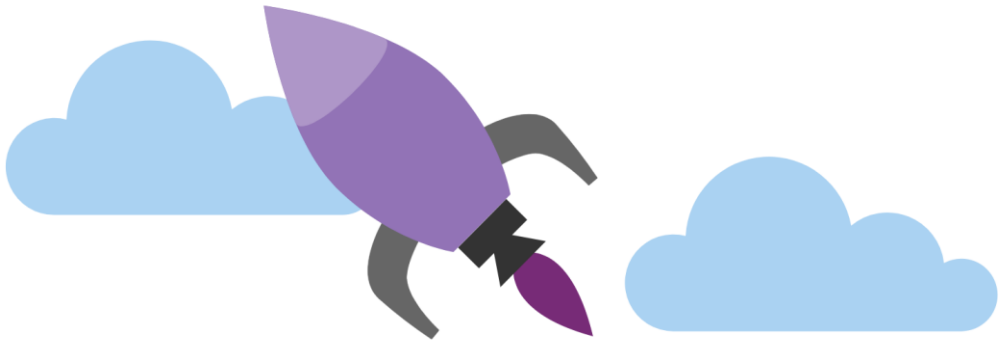
.NET Core 2.0 Launch Event, Tehran-IRAN, 2017, 4 Sep

Amin Mesbahi

About me

Amin Mesbahi

- Software Architect, Consultant and Instructor
- Senior Program Manager
- High Concurrency Solutions
- Mission-Critical Software

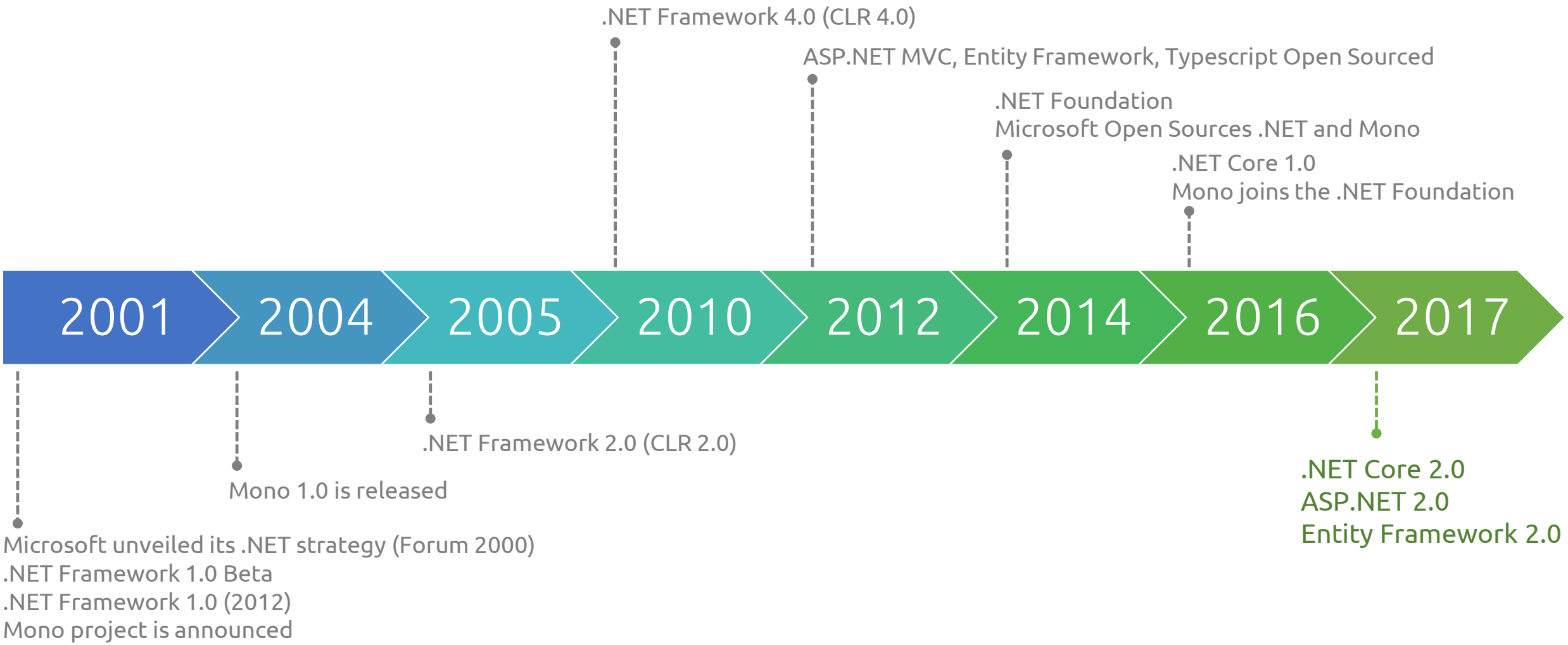


Email: amin@Mesbahi.net
Website: Mesbahi.net
LinkedIn: ir.linkedin.com/in/aminmesbahi

Agenda

- Story of .NET
- Introduction to .NET Core
- .NET Framework vs .NET Core
- .NET Core Architecture
- What has changed about ASP.NET and Entity Framework?
- When to use .NET Core? (or not)
- How to migrate existing code to .NET Core
- Some Good Resources
- Case Study

History of a Framework



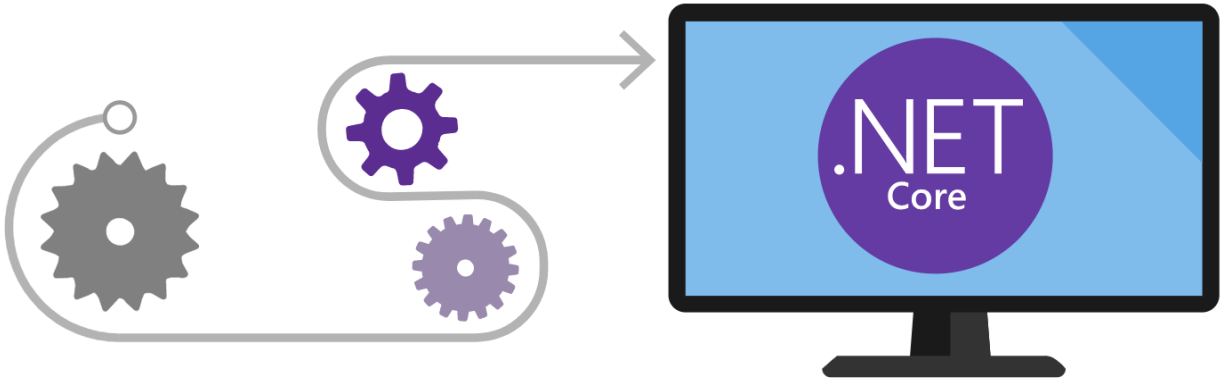
Introducing a new Generation

- Cross-platform
- Open source
- Microservices architecture
- Containers
- Modern Architecture
- Modular Design
- Various development tools
- A need for high-performance and scalable systems
- A need for side by side of .NET versions per application level

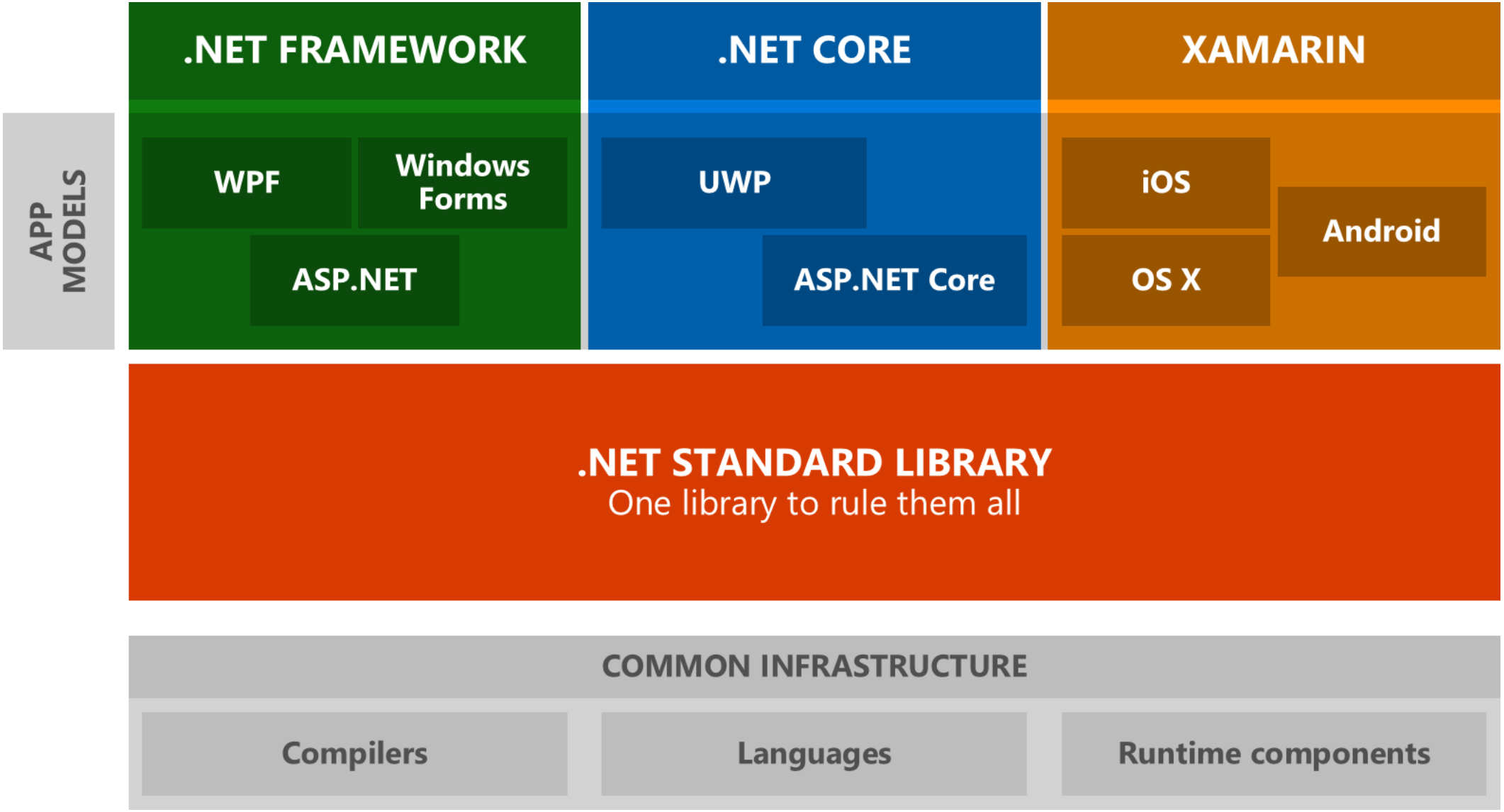
.NET Framework vs .NET Core

.NET Core	.NET Framework
You need training, searching and developing	Develop easier for legacy teams
Windows, macOS, and Linux on AMD64, x86, and ARM	Windows-only, PC-only, deeply tied to IIS
Modular	A whole framework
UWP, ASP.NET Core, Razor Pages, CLI	WPF, Windows Forms, ASP.NET (WebForm, MVC, Pages)
.NET Core is much faster High-performance and scalable system without UI	Speed is not an important concern
You are using Docker containers	You run your app in old fashion
You don't need SignalR, WCF Client Library, WorkFlow	You need your current code and 3rd Party Libraries

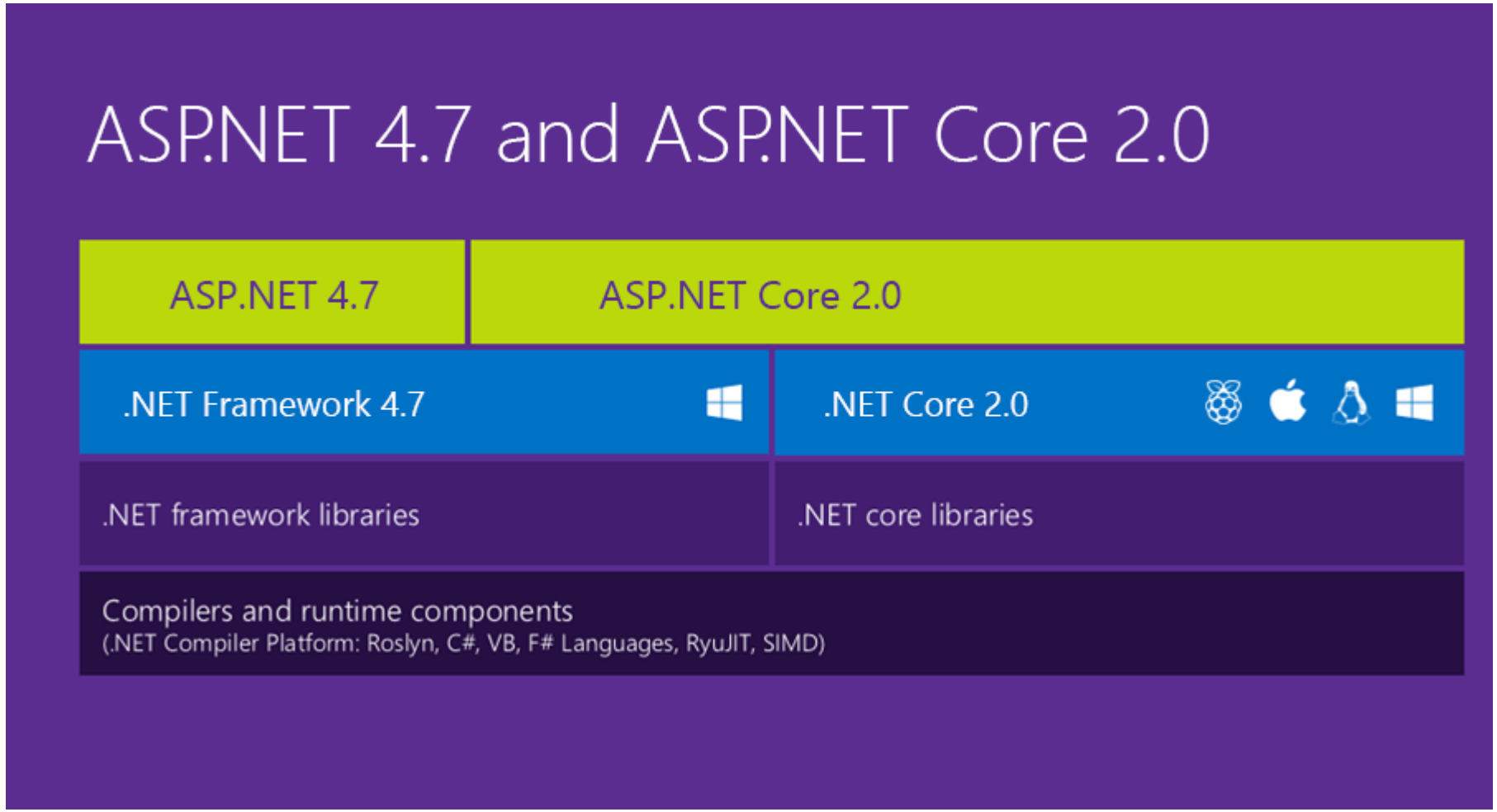
.NET Core is the future of .NET at Microsoft. It is going to replace all the different slightly incompatible independent implementations of .NET inside Microsoft.



.NET Framework vs .NET Core



.NET Core Architecture



Deployment Types

- **Cross-platform:** Console apps can run on Windows, OS X, and Linux.

- **Native compilation:** This combines the benefits of a managed app with the performance of a native C/C++ app.

.NET Core Components:

Common Language Runtime: which in .NET Core is named CoreCLR

CoreFX: a modular collection of libraries

The set of available APIs in .NET Standard from 13k in .NET Standard 1.6 increased to 32k in .NET Standard 2.0.

ASP.NET Core Overview:

ASP.NET Core is a new open-source and cross-platform framework for building modern cloud based internet connected applications, such as:

- Web apps
- IoT apps
- Mobile backends

ASP.NET Core 2.0 runs on both .NET Framework 4.6.1 and .NET Core 2.0, so you will need to update your target framework in your project to netcoreapp2.0.

Architected to provide an optimized development framework for Cloud or run On-Premise apps.

.NET Core 2.0 Templates:

- Console Application
- Class library
- Unit Test Project
- xUnit Test Project
- ASP.NET Core Empty
- ASP.NET Core Web App (Model-View-Controller) MVC
- ASP.NET Core Web App Razor Pages (new in 2.0)
- ASP.NET Core with Angular SPA
- ASP.NET Core with React.js SPA
- ASP.NET Core with React.js and Redux SPA
- ASP.NET Core Web API REST

Community based templates...

SignalR will come in 2.1

ASP.NET Core

ASP.NET Core is no longer based on System.Web.dll. It is based on a set of granular and well factored NuGet packages.

Benefits of smaller app surface area:

- Tighter security
- Reduced servicing
- Improved performance
- Decreased costs in a pay-for-what-you-use model

ASP.NET Core Anatomy

- Everything starts from Program.cs, Main Method
- ASP.NET Core apps require a Startup class
- No more Global.asax
- No more Web.Config requirement

ASP.NET Core Anatomy – Program.cs

ASP.NET Core 1.x

```
public class Program
{
  public static void Main(string[] args)
  {
    var host = new WebHostBuilder()
      .UseKestrel()
      .UseContentRoot(Directory.GetCurrentDirectory())
      .UseIISIntegration()
      .UseStartup<Startup>()
      .Build();

    host.Run();
  }
}
```

The CreateDefaultBuilder does what the old one does leaving you only to have to specify the startup. Webhost.cs

ASP.NET Core 2.x

```
public class Program
{
  public static void Main(string[] args)
  {
    BuildWebHost(args).Run();
  }

  public static IWebHost BuildWebHost(string[] args) =>
      WebHost.CreateDefaultBuilder(args)
          .UseStartup<Startup>()
          .Build();
}
```

ASP.NET Core Anatomy – Program.cs

The Program.BuildWebHost method is provided by convention so that tools, like EF migrations, can inspect the WebHost for the app without starting it.

You should not do anything in the BuildWebHost method other than building the WebHost. We used an expression-bodied method in the templates to help indicate that this method shouldn't be used for anything other than creating an IWebHost.

ASP.NET Core Anatomy – Program.cs

The Program.BuildWebHost method is provided by convention so that tools, like EF migrations, can inspect the WebHost for the app without starting it.

You should not do anything in the BuildWebHost method other than building the WebHost.

2.x

```
public static void Main(string[] args)
    {
        WebHost.Start(async context => await context.Response.WriteAsync("Hello World!"))
                .WaitForShutdown();
    }
```

ASP.NET Core, BuildWebHost

2.x

- WebHostBuilderContext added a to WebHostBuilder. WebHostBuilderContext allows these services to be configured earlier, and be available in more places:

```
public static IWebHost BuildWebHost(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .ConfigureLogging((context, logging) =>
            logging.AddMyLogger(context.Configuration["MyLogger:Configuration"]))
        .Build();
```

ASP.NET Core Startup class

- The request pipeline is configured by adding middleware components to an IApplicationBuilder instance that is provided by dependency injection

```
public void ConfigureServices(IServiceCollection services)
```

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env,
 ILoggerFactory loggerFactory)
```

ASP.NET Core Startup class

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env,
 ILoggerFactory loggerFactory)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }
    app.UseStaticFiles();
    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });
```

- Each Use extension method adds a middleware component to the request pipeline.

ASP.NET Core Startup class

2.x

- There are several services that should always be available in ASP.NET Core like IConfiguration, ILogger (and ILoggerFactory), and IHostingEnvironment.

  In 2.0 an IConfiguration object will now always be added to the IoC container, this means that you can accept IConfiguration in your controller or other types activated with DI just like you can with ILogger and IHostingEnvironment

ASP.NET Core Logging

2.x

There are three main differences in the way that Logging can be used in 2.0:

1. Providers can be registered and picked up from DI instead of being registered with ILoggerFactory, allowing them to consume other services easily.
2. It is now idiomatic to configure Logging in your Program.cs. This is partly for the same reason that configuration is now a core service.
3. The log filtering feature that was previously implemented by a wrapping LoggerFactory is now a feature of the default LoggerFactory, and is wired up to the registered configuration object.

ASP.NET Core Logging

2.x

Instead of accepting an ILoggerFactory in Configure method in Startup.cs

```
public class Program
{
    public static void Main(string[] args)
    {
        BuildWebHost(args).Run();
    }
    public static IWebHost BuildWebHost(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>()
            .ConfigureLogging((hostingContext, logging) =>
            {
                logging.AddConfiguration(hostingContext.Configuration.GetSection("Logging"));
                logging.AddConsole();
                logging.AddDebug();
            })
            .Build();
}
```

Kestrel Hardening

2.x

The Kestrel web server has new features that make it more suitable as an Internet-facing server (KestrelServerOptions class's new Limits property)

- Maximum client connections
- Maximum request body size
- Minimum request body data rate

WebListener

1.1

WebListener is a server that runs directly on top of the Windows Http Server API. WebListener gives you the option to take advantage of Windows specific features, like support for Windows authentication, port sharing, HTTPS with SNI, HTTP/2 over TLS (Windows 10), direct file transmission, and response caching WebSockets (Windows 8). This may be advantageous for you if you want to bundle an ASP.NET Core microservice in a Windows container that takes advantage of these Windows features.

2.x

The packages Microsoft.AspNetCore.Server.WebListener and Microsoft.Net.Http.Server have been merged into a new package Microsoft.AspNetCore.Server.HttpSys. The namespaces have been updated to match.

Automatic Page and View compilation on publish

2.x

Razor page and view compilation is enabled during publish by default, reducing the publish output size and application startup time. This means that your razor pages and views will get published with your app as a compiled assembly instead of being published as .cshtml source files that get compiled at runtime. If you want to disable view pre-compilation, then you can set a property in your csproj like this:

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp2.0</TargetFramework>
    <MvcRazorCompileOnPublish>false</MvcRazorCompileOnPublish>
  </PropertyGroup>
```

Tag Helper components

2.x

Tag helper components are responsible for generating or modifying a specific piece of HTML. They are registered as services and optionally executed by TagHelpers.

```
public class JavaScriptSnippetTagHelperComponent : TagHelperComponent
{
    public override int Order => 100;

    public override void Process(TagHelperContext context, TagHelperOutput output)
    {
        if (string.Equals(context.TagName, "head", StringComparison.OrdinalIgnoreCase))
        {
            output.PostContent.AppendHtml("<script src='myscript.js'></script>");
        }
    }
}
```

IHostedServices

2.x

If you register an IHostedService then ASP.NET Core will call the Start() and Stop() methods of your type during application start and stop respectively. Specifically, start is called after the server has started and IApplicationLifetime.ApplicationStarted is triggered.

Today we only use hosted services in SignalR, but we have discussed using it for things like:

- An implementation of QueueBackgroundWorkItem that allows a task to be executed on a background thread
- Processing messages from a message queue in the background of a web app while sharing common services such as ILogger

IHostingStartup

2.x

When you publish to an Azure App Service and enable Application Insights you get log messages and other telemetry "for free", meaning that you don't have to add any code to your application to make it work. This automatic light-up feature is possible because of the IHostingStartup interface and the associated logic in the ASP.NET Core hosting layer.

The IHostingStartup interface defines a single method: void Configure(IWebHostBuilder builder);. This method is called while the WebHost is being built in the Program.cs of your ASP.NET Core application and allows code to setup anything that can be configured on a WebHostBuidler, including default services and loggers which is how Application Insights works.

Improved TempData support

In ASP.NET Core 1.1 and higher, you can use the cookie-based TempData provider to store a user's TempData in a cookie.

2.x

- The cookie TempData provider is now the default TempData provider. This means you no longer need to setup session support to make use of the TempData features
- You can now attribute properties on your controllers and page models with the TempDataAttribute to indicate that the property should be backed by TempData. Set the property to add a value to TempData, or read from the property to read from TempData.

Introducing Razor Pages 2.x

a new coding paradigm that makes writing page-focused scenarios easier and simpler than our current Model-View-Controller architecture. Razor Pages are a page-first structure that allow you to focus on the user-interface and simplify the server-side experience by writing PageModel objects.

It's a replacement for old-fashion ASP.NET Pages framework (introduced with WebMatrix)

Introducing Razor Pages

2.x

```
@page

@functions {

  public string FormatDate(DateTime theTime) {
    return theTime.ToString("d");
  }
}
<html>
    <body>
        <h2>The server-local time now is:</h2>

        <p>@FormatDate(DateTime.Now)</p>
    </body>
</html>
```

- The AddMvc and UseMvc configuration calls in your Startup class also activate the Razor Pages feature. You can start writing a Razor Page by placing a new cshtml file called Now.cshtml in the Pages/ top-level folder of your application.

Model in Razor Pages

2.x

```
namespace Seminar.Pages
{
  public class NowModel : PageModel
  {
    private IFileProvider _FileProvider;

    public NowModel(PhysicalFileProvider fileProvider)
    {
      _FileProvider = fileProvider;
      LastModified = _FileProvider.GetFileInfo("Pages/Now.cshtml").LastModified.LocalDateTime;
    }
    public DateTime LastModified { get; set; }

    public void OnGet() { }
  }
}
```

Usage

```
@page
@model Seminar.Pages.NowModel
```

ASP.NET Core 2.0 other new features

- Razor Support for C# 7.1
- DbContext Pooling with Entity Framework Core 2.0
- Monitor and Profile with No Code Changes and Application Insights
- ASP.NET Core Metapackage/Runtime Store
- Media type suffixes
- PERFORMANCE ENHANCEMENTS ☺

Entity Framework Core 2.0

.NET Standard 2.0

EF Core now targets the new .NET Standard 2.0. The latter defines a shared surface area of over 32,000 APIs that works across .NET Framework, .NET Core, Mono, Xamarin and soon, the Universal Windows Platform. With .NET Standard 2.0, developers can reuse their code and skills on a wide range of platforms, application types and devices.

Entity Framework Core 2.0

Improved LINQ translation

increased the number of patterns that can be translated to SQL, so many queries that triggered client-side evaluation in previous versions will no longer do it in 2.0

Like query operator

You can now use EF.Functions.Like() in a LINQ query and it will be translated to LIKE in SQL or evaluated in memory if necessary. E.g. the following query:

```
var customers =
    from c in context.Customers
    where EF.Functions.Like(c.Name, "a%");
    select c;
```

```
1   SELECT [c].[Id], [c].[Name]
2   FROM [Customers] AS [c]
3   WHERE [c].[Name] LIKE N'a%';
```

Entity Framework Core 2.0

Owned entities and Table Splitting

You can now define “owned” or “child” entities which group properties within other entities, very similar to how complex types used to work in EF6, but with the ability to contain reference navigation properties.

```
public class Customer
{
    public int Id { get; set; }
    public string Name {get; set;}
    public PhysicalAddress Address { get; set; }
}
public class PhysicalAddress
{
    public string StreetAddress { get; set; }
    public Location Location { get; set; }
}
```

```
modelBuilder.Entity<Customer>()
    .OwnsOne(c => c.Address);
```

Entity Framework Core 2.0

Global query filters

You can now specify filters in the model that are applied automatically to all entities of a type in all queries executed on the DbContext. E.g. given this code in OnModelCreating:

```
modelBuilder.Entity<Post>()
    .HasQueryFilter(p => !p.IsDeleted);
```

DbContext Pooling

Many ASP.NET Core applications can now obtain a performance boost by configuring the service registration of their DbContext types to use a pool of pre-created instances, avoiding the cost of creating new instance for every request

Entity Framework Core 2.0

String interpolation in raw SQL methods

The following SQL query using C# string interpolation syntax now gets correctly parameterized:

```
var city = "Redmond";

using (var context = CreateContext())
{
    context.Customers.FromSql($@"
        SELECT *
        FROM Customers
        WHERE City = {city}");
}
```

```
SELECT *
FROM Customers
WHERE City = @p0
```

Entity Framework Core 2.0

- explicitly compiled queries
- self-contained entity configurations in code first
- database scalar function mapping

Entity Framework Core 2.0 (dark side)

- Microsoft has indicated that grouping support is being planned for EF Core 2.1
- EF Core doesn't support complex types, but instead has "owned" or "child" types.
- lazy loading is being delayed until EF Core 2.1
- TPT (Table Per Type Inheritance) isn't supported at all in EF Core. There is a backlog item for it, but no specific plans to implement it
- TPC (Table per Class) also not supported by EF Core, there doesn't seem to be a lot of call for TPC.

Entity Framework Core 2.0 (dark side)

- Another missing feature is full support for stored procedures
- Working with spatial data is still not supported by EF Core. It is on the road map as a high priority item, but not tied to a specific release yet.

When to use .NET Core? (or not)

- You will use it even you don't like it! But not now…
- Measure your agility and geekiness, before starting what didn't get matured!
- Did you plan for the feature or you need only a new old!
- Check your third-party tools compatibility

How to migrate existing code to .NET Core

- Develop from scratch
- Develop incrementally if your architecture allow

Some Good Resources

- docs.microsoft.com samples
- MS .NET Core Guide
- Serenity, bit-framework, ASP-NET-Core-Boilerplate,
- MusicStore, eShopOnWeb
- Orchard, CloudScribe

Case Study, Bama.ir

bama.ir

پروفایل | ثبت نام

بیایید بسپارید بدانید باما

باما › خودرو

خرید خودرو | فروش خودرو؟ کلیک کنید

4 0 7 0 3 خودرو

برند | از سال | تا سال | از کارکرد | تا کارکرد | به ترتیب ...

مدل | از قیمت | تا قیمت | پرداخت | عکس دار | جستجو

استان

تهران (19090)

اصفهان (2529)

البرز (1176)

خراسان رضوی (693)

فارس (682)

مازندران (582)

آذربایجان شرقی (473)

دیگر استان ها...

شاسی

سواری (19221)

شاسی بلند (7297)

وانت (511)

کوپه (239)

2014، بنز، S500 [1396/05/03]

تهران، سعادت آباد

سرمه ای، بدون رنگ، 1 edition داخل مشکی کرم بسیار سالم در حد صفر سور....

کارکرد 20,000 کیلومتر

2,500,000,000 تومان

2015، مازراتی، گیبلی [18 ساعت پیش]

نمایشگاه

تهران، زعفرانیه

سفید، بدون رنگ، گیبلی Q4 S ، منحصر به فرد ،صفر خشک، برای کسب اطلاع....

کارکرد صفر کیلومتر

2,200,000,000 تومان

2015، لامبورگینی، اونتادور [19 ساعت پیش]

تهران،

گذر موقت، زرد، بدون رنگ، منحصر به فرد ! شاهکار کمپانی مطرح لامبورگینی را با....

Q&A

Thanks for your time ☺

amin@mesbahi.net