

.NET Core: Developing Cross-Platform Web Apps with ASP.NET Core – Workshop*PLUS*

< Engineer Name >

Customer Engineer

v3.1

Conditions and Terms of Use

Microsoft Confidential

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet Web site references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

Copyright and Trademarks

© 2016 Microsoft Corporation. All rights reserved.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see Use of Microsoft Copyrighted Content at

<http://www.microsoft.com/en-us/legal/intellectualproperty/permissions/default.aspx>

Active Directory, Azure, IntelliSense, Internet Explorer, Microsoft, Microsoft Corporate Logo, Silverlight, SharePoint, SQL Server, Visual Basic, Visual Studio, Windows, Windows Server, and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

Module 2: ASP.NET Overview

Module Overview

Module 2: Overview

Section 1: Modern Web

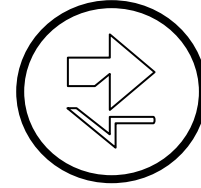
Lesson: What is ASP.NET Core

ASP.NET Core is an open-source and cross-platform framework for building modern cloud based internet connected applications, such as web apps, IoT apps and mobile backends.

ASP.NET Core and the Modern Web



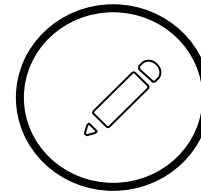
Totally Modular



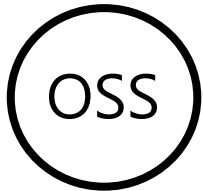
Faster Development Cycle



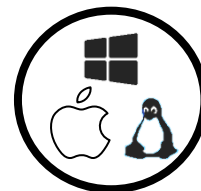
Seamless transition
from on-premises to cloud



Choose your Editors
and Tools



Open Source
with Contributions



Cross-Platform



Fast

Demo: ASP.NET Core and Visual Studio 2019

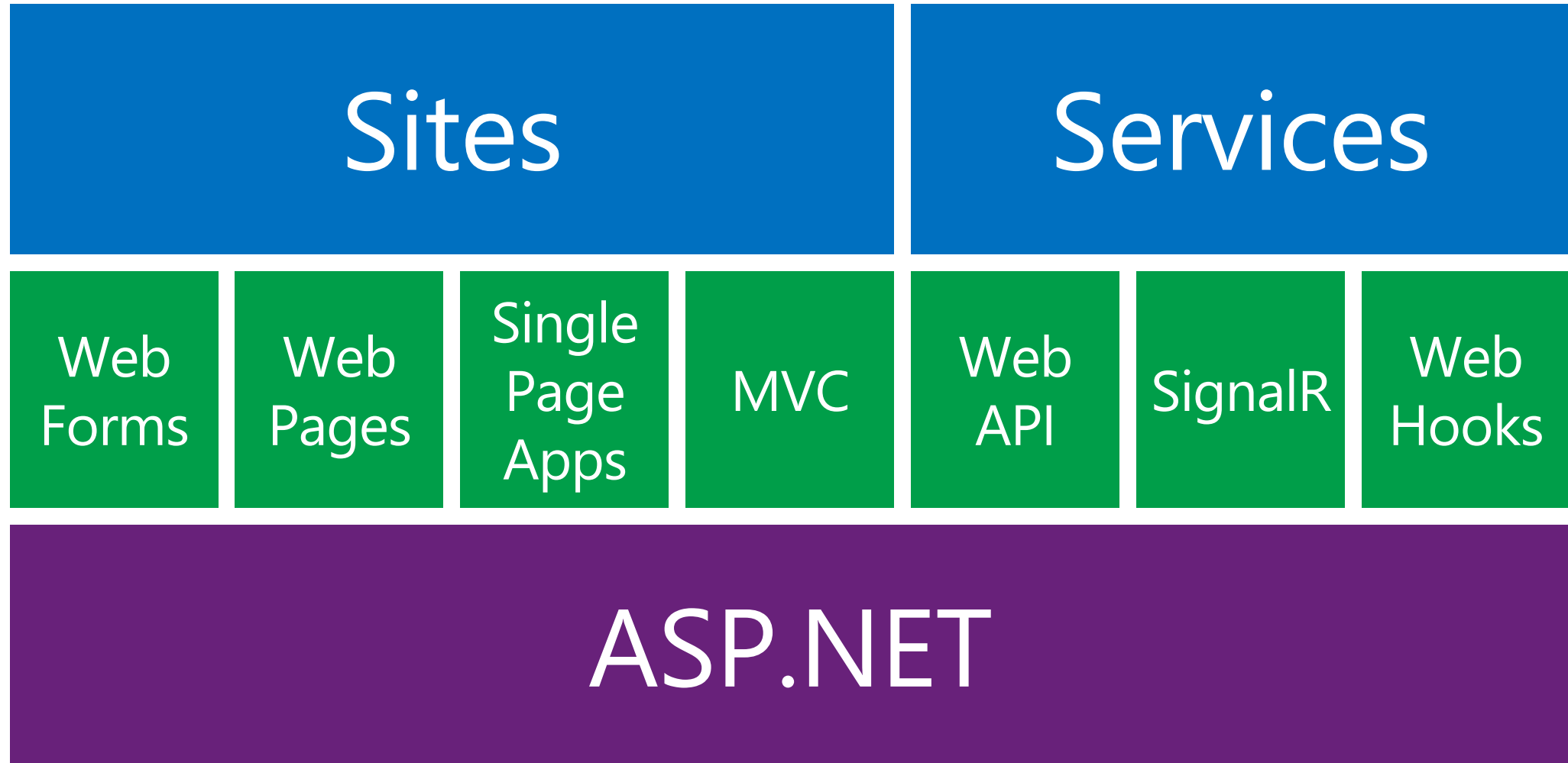
Module 2: Overview

Section 1: Modern Web

Lesson: One ASP.NET

History of ASP

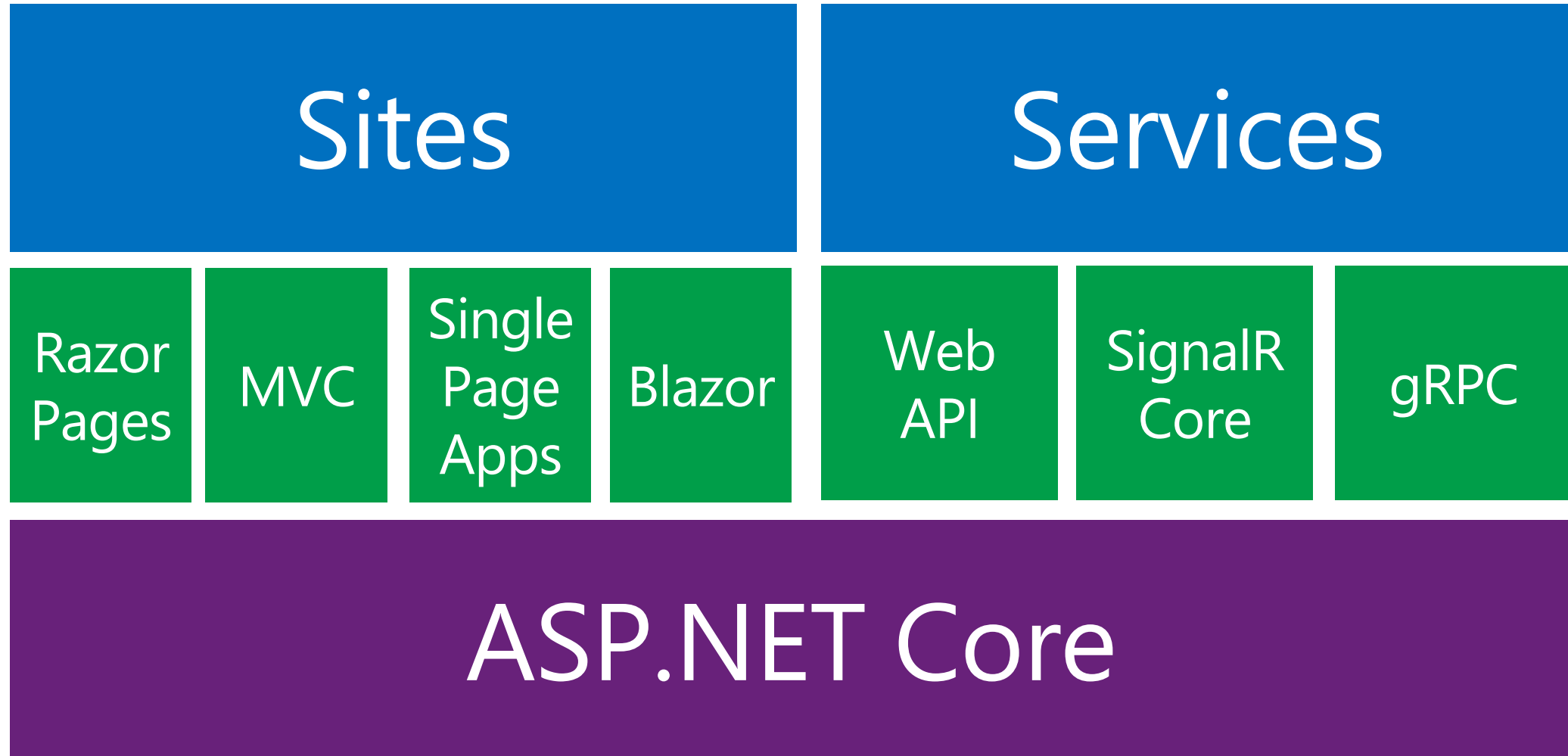
- 1996 – Active Server Pages
- 2002 – ASP.NET
- 2008 – ASP.NET MVC
- 2010 – ASP.NET Web Pages
- 2012 – ASP.NET Web API, SignalR
- 2016 – ASP.NET Core 1.0 & .NET Core 1.0
- 2017 – ASP.NET Core 2.0 & .NET Core 2.0
- 2018 – ASP.NET Core 2.1, ASP.NET Core SignalR & .NET Core 2.1
- 2018 – ASP.NET Core 2.2 & .NET Core 2.2
- **2019 – ASP.NET Core 3.1 & .NET Core 3.1**
- 2020 – ASP.NET Core 5 & .NET 5 (Just released – 10th of November 2020)



Commonalities

- All programming models have the same Microsoft ASP.NET
 - Authentication/Authorization/Membership
 - Output Caching, Session State, and Configuration
 - AJAX, Deployment, etc.
- All programming models are fully supported and will continue to be supported
- All programming models solve real problems

ASP.NET Core




ASP.NET Core Project System


Create a new ASP.NET Core web application

.NET Core


ASP.NET Core 3.1

 **Empty**


An empty project template for creating an ASP.NET Core application. This template does not have any content in it.

 **API**


A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.

 **Web Application**


A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.

 **Web Application (Model-View-Controller)**

A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

 **Angular**

A project template for creating an ASP.NET Core application with Angular

 **React.js**

Authentication

No Authentication

[Change](#)

Advanced

☒ **Configure for HTTPS**

☐ **Enable Docker Support**

(Requires [Docker Desktop](#))

Linux

Author: Microsoft

Source: .NET Core 3.1.3

[Get additional project templates](#)

Back

Create

Demo: One ASP.NET

Which One is Right for Me?

ASP.NET 4.x	ASP.NET Core
Build for Windows	Build for Windows, macOS, or Linux
Use Web Forms , SignalR , MVC , Web API , WebHooks , or Web Pages	Razor Pages is the recommended approach to create a Web UI as of ASP.NET Core 2.x. See also MVC , Web API , Blazor , gRPC , and SignalR .
One version per machine	Multiple versions per machine
Develop with Visual Studio using C#, VB, or F#	Develop with Visual Studio, Visual Studio for Mac , or Visual Studio Code using C# or F#
Good performance	Higher performance than ASP.NET 4.x
Use .NET Framework runtime	Use .NET Core runtime

- <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/choose-aspnet-framework>

ASP.NET 4.x vs. ASP.NET Core

MSBuild /CodeDOM > csc.exe	Compilation	.Net CLI (Roslyn)
Loose, GAC, NuGet	Libraries	NuGet
FCL, GAC, NuGet	Application Frameworks	NuGet
IIS	Web Server	IIS, HTTP.SYS, Kestrel
.NET BCL and FCL	Platform Libraries	.NET BCL and FCL; .NET on NuGet
.NET CLR	Runtime	.NET CLR; .NET Core CLR
IIS: WebEngine4.dll; EXE: OS	Runtime Loader	.Net CLI
Windows	Operating System	Windows, OSX, Linux

Module 2: Overview

Section 3: ASP.NET Core

Lesson: ASP.NET Core Projects

ASP.NET Core Project File

- ***.csproj**
 - Simplified project file
 - Automatically includes all source files in/under the folder containing project.json
- All project folder files shown in Solution Explorer
 - Visual Studio automatically monitors the ASP.NET Core project directory files

ASP.NET Core Project File Contents

The screenshot displays the Visual Studio IDE with the `AspNetCoreApp1.csproj` file open in the editor. The code defines an ASP.NET Core project using the `Microsoft.NET.Sdk.Web` SDK. The `TargetFramework` is set to `netcoreapp3.1`. The Solution Explorer on the right shows the project structure, including a red arrow pointing to the `Microsoft.AspNetCore.App` framework reference.

```
1 <Project Sdk="Microsoft.NET.Sdk.Web">
2
3   <PropertyGroup>
4     <TargetFramework>netcoreapp3.1</TargetFramework>
5   </PropertyGroup>
6
7
8
9 </Project>
10
```

Solution Explorer:

- Solution 'AspNetCoreApp1' (1 of 1 project)
 - AspNetCoreApp1
 - Connected Services
 - Dependencies
 - Analyzers
 - Frameworks
 - Microsoft.AspNetCore.App
 - Microsoft.NETCore.App
 - Properties
 - wwwroot
 - Pages
 - appsettings.json
 - Program.cs
 - Startup.cs

ASP.NET Core Project File Explained

- Although the .NET Core is built on top of NuGet **packages**, in .NET Core 3.x has its **shared framework** location
- A shared framework provides a set of libraries at a central location so they can be used by different applications.
- You don't have to explicitly include the **metapackage** in the .csproj file
- So what is the difference between: package, metapackage and framework?

Packages, Metapackages & Frameworks

Packages

.NET Core is split into a set of packages that provide primitives, higher-level data types, app composition types, and common utilities. Each of these packages represents a single assembly of the same name. For example, the [System.Runtime package](#) contains System.Runtime

Metapackage

A metapackage is a NuGet package convention for describing a set of packages that are meaningful together. A metapackage represents this set of packages by making them dependencies. The metapackage can optionally establish a framework for the set of packages by specifying a framework.

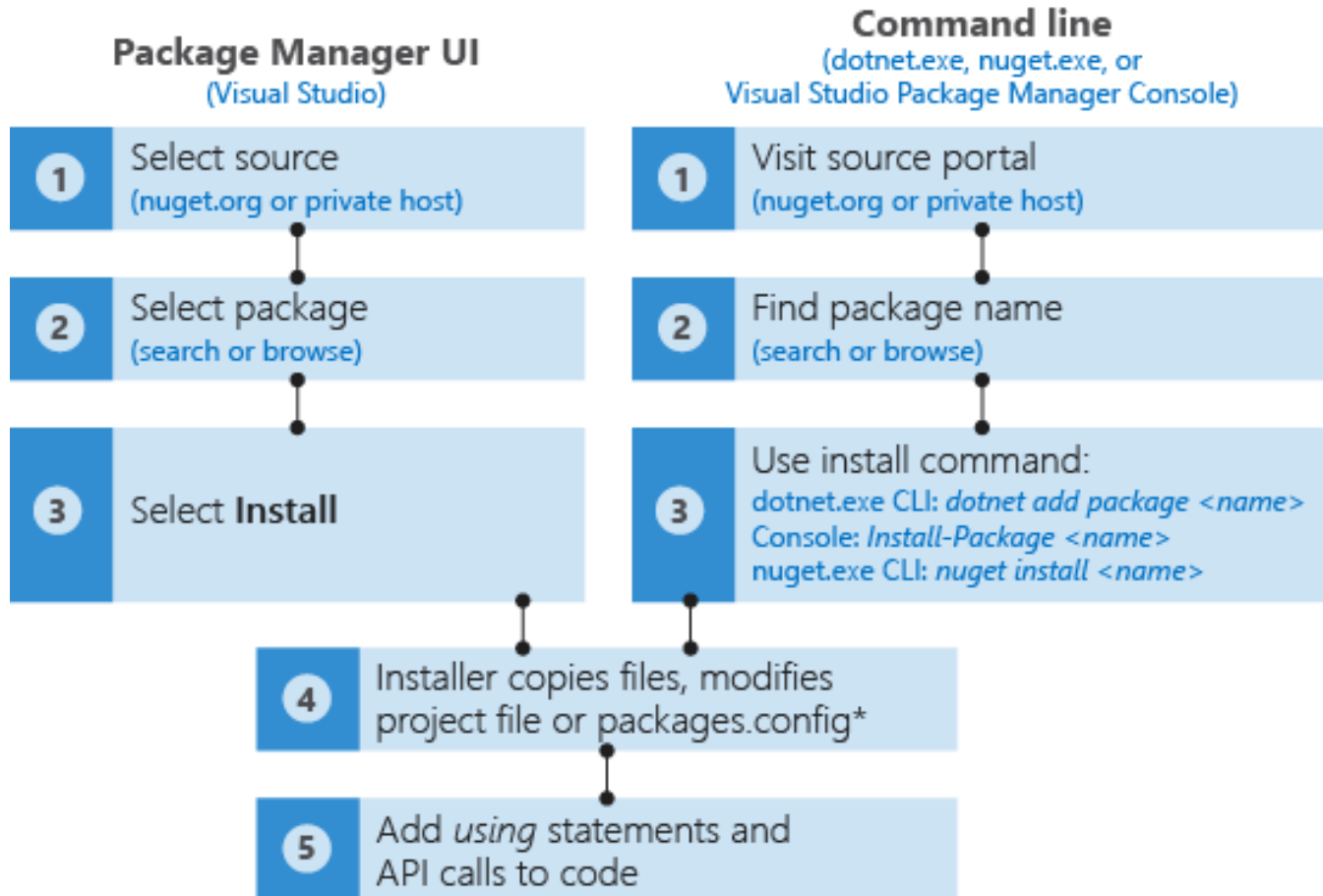
Framework

.NET Core packages each support a set of runtime frameworks. Frameworks describe an available API set (and potentially other characteristics) that you can rely on when you target a given framework. They are versioned as new APIs are added.

What is NuGet

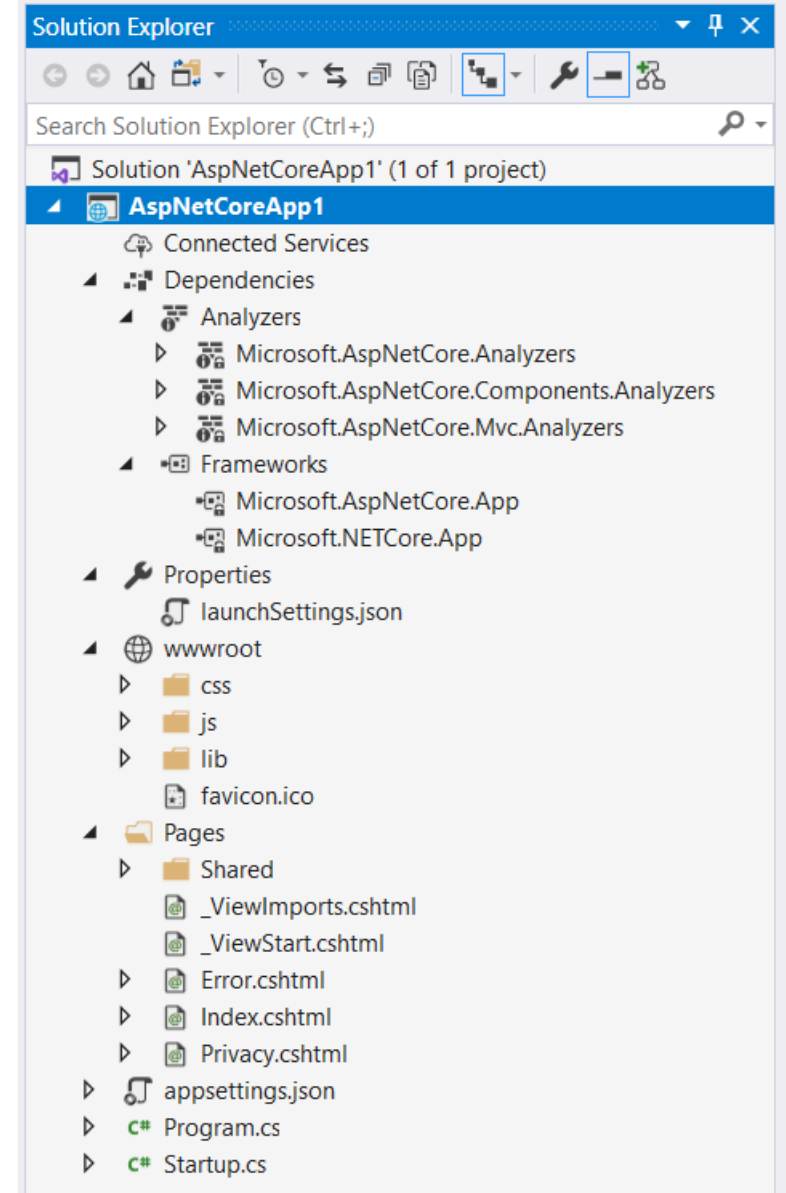
- NuGet is the package manager for .NET. The NuGet client tools provide the ability to produce and consume packages. The NuGet Gallery is the central package repository used by all package authors and consumers.
- An essential tool for any modern development platform is a mechanism through which developers can create, share, and consume useful code. Often such code is bundled into "packages" that contain compiled code (as DLLs) along with other content needed in the projects that consume these packages.
- Packages can be hosted on different sources
 - <https://www.nuget.org> – requires internet connection
 - Private source – can be online or offline

How to consume NuGet Packages



ASP.NET Core Project Layout

- wwwroot folder
 - Static file placeholder
- Dependencies
 - Analyzers
 - NuGet packages
 - No direct DLL references
- Configuration through appsettings.json.
- No web.config unless app is hosted in IIS or IIS Express



Module 2: Overview

Section 3: ASP.NET Core

Lesson: Configurations

Configuration files are completely **pluggable**

The default JSON format can be replaced with an XML/.ini file or even take settings from a SQL Server database

ASP.NET Core Configuration Sources

App configuration in ASP.NET Core is based on key-value pairs established by configuration providers. Configuration providers read configuration data into key-value pairs from a variety of configuration sources:

- Azure Key Vault
- Command-line arguments
- Custom providers (installed or created)
- Directory files
- Environment variables
- In-memory .NET objects
- Settings files

Generic Host Default Configuration

What's a host?

A host is an object that encapsulates an app's resources, such as:

- Dependency injection (DI)
 - Logging
 - Configuration
 - **IHostedService** implementations
-
- Host configuration is provided from:
 - Environment variables prefixed with **DOTNET_** (for example, **DOTNET_ENVIRONMENT**) using the [Environment Variables Configuration Provider](#).
The prefix (**DOTNET_**) is stripped when the configuration key-value pairs are loaded.
 - Command-line arguments using the Command-line Configuration Provider.

ASP.NET Core Web Host Default Configuration

- Web Host default configuration is established (**ConfigureWebHostDefaults**):
 - Kestrel is used as the web server and configured using the app's configuration providers.
 - Add Host Filtering Middleware.
 - Add Forwarded Headers Middleware if the `ASPNETCORE_FORWARDEDHEADERS_ENABLED` environment variable is set to **true**.
 - Enable IIS integration.
- App configuration is provided from:
 - *appsettings.json* using the [File Configuration Provider](#).
 - *appsettings.{Environment}.json* using the [File Configuration Provider](#).
 - [Secret Manager](#) when the app runs in the **Development** environment using the entry assembly.
 - Environment variables using the [Environment Variables Configuration Provider](#).
 - Command-line arguments using the [Command-line Configuration Provider](#).

File Configuration Provider

- [INI Configuration Provider](#)
- [JSON Configuration Provider](#)
- [XML Configuration Provider](#)

Order configuration providers in code to suit the priorities for the underlying configuration sources that the app requires.

The last provider in the order is to override its predecessor.

```
public static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .ConfigureAppConfiguration((_, config) =>
        {
            config.AddInMemoryCollection(arrayDict);
            config.AddIniFile("app_configs.ini", optional: false, reloadOnChange: false);
            config.AddJsonFile("app_configs.json", optional: false, reloadOnChange: false);
            config.AddXmlFile("app_configs.xml", optional: false, reloadOnChange: false);
            config.AddCommandLine(args);
        })
```

Demo:

ASP.NET Core Project and Configuration
System

Module 2: Overview

Section 3: ASP.NET Core

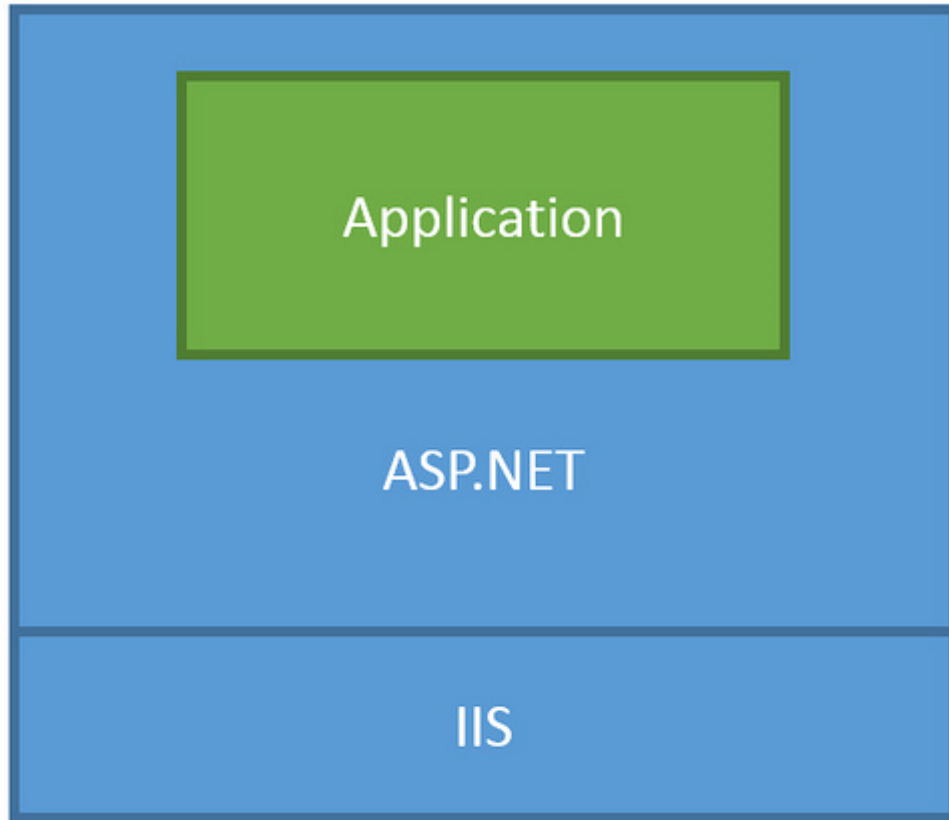
Lesson: Middleware

Middleware

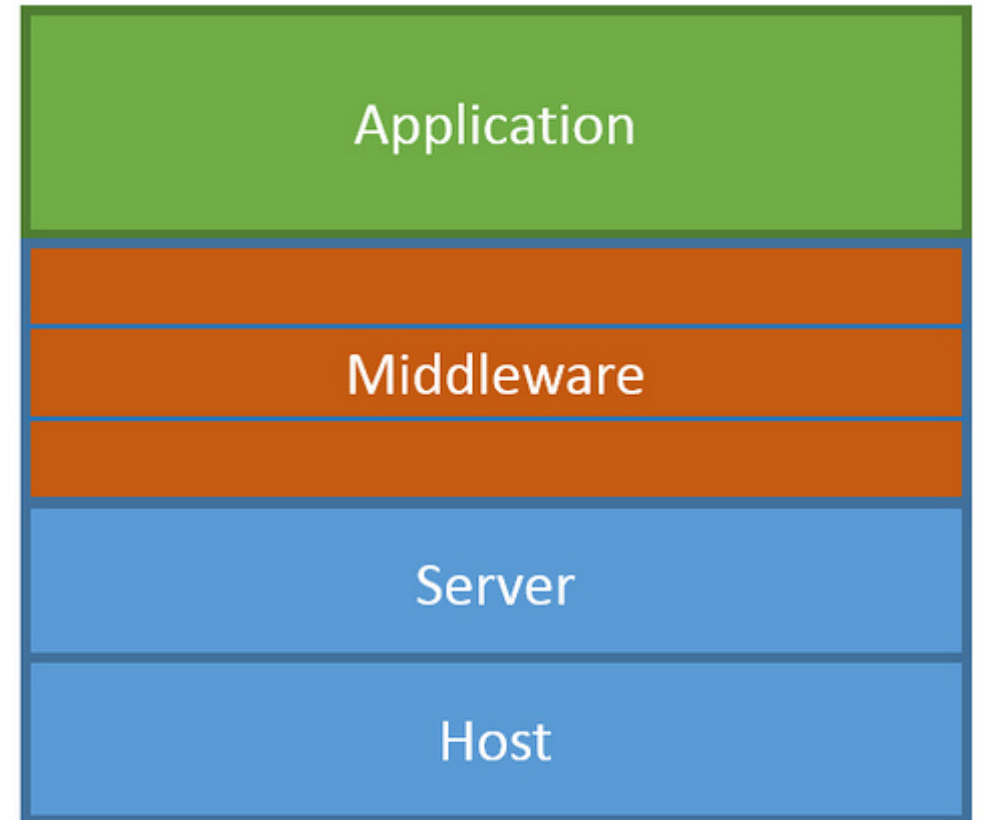
- Middleware is software that's assembled into an app pipeline to handle requests and responses. Each component:
 - Chooses whether to pass the request to the next component in the pipeline.
 - Can perform work before and after the next component in the pipeline.
- Integrated support by ASP.NET Core
- Wired up in **Configure** method of **Startup** class
- Either invokes the next component in chain or short-circuits it
- **Run**, **Map**, and **Use** extension methods
 - MVC engine is configured in Request Pipeline through Use extension method
- Implemented in-line as anonymous method, or through reusable class
- Order of **Use[Middleware]** statements in application's Configure method is very important

ASP.NET Core Middleware

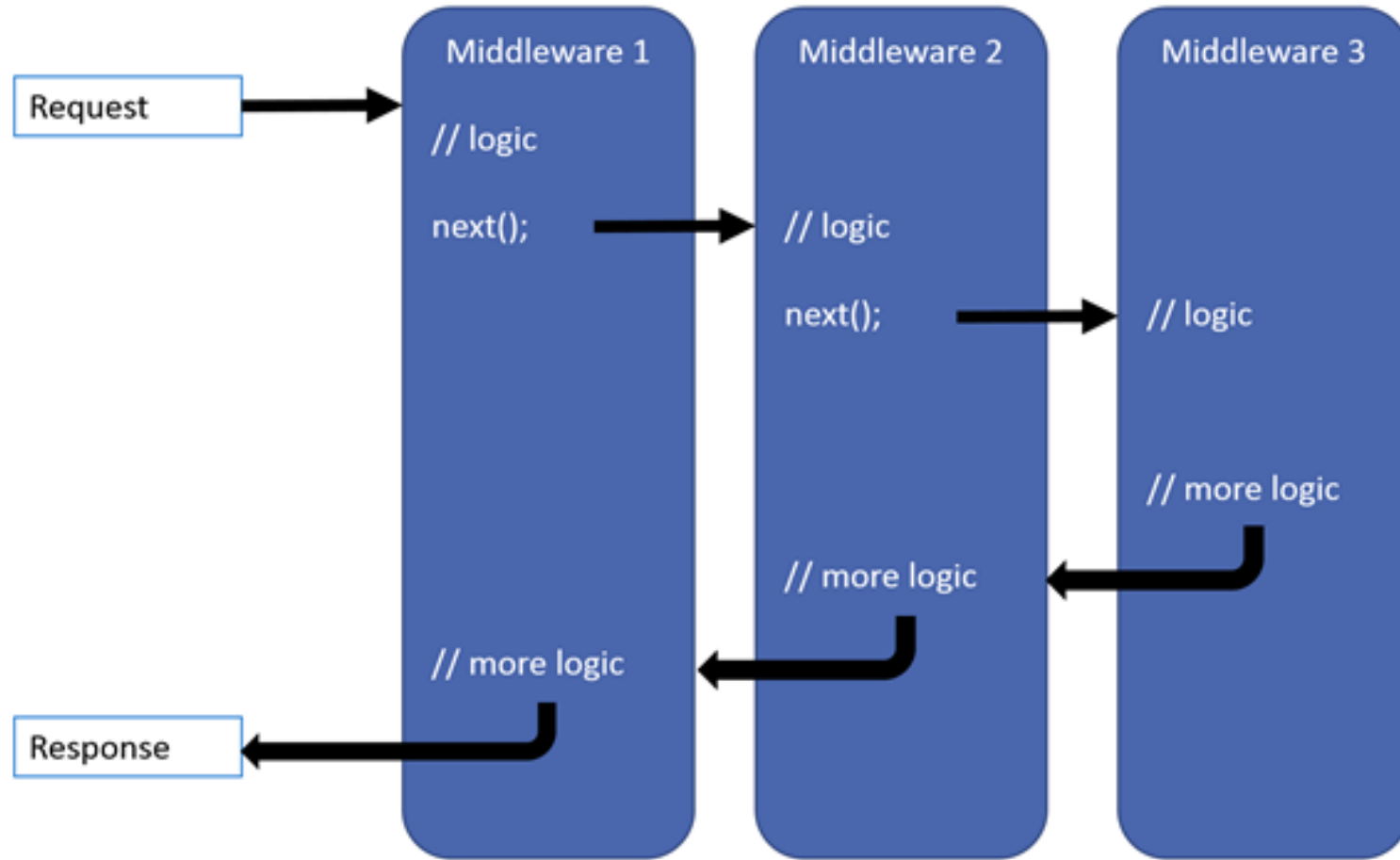
Traditional ASP.NET Application Model



ASP.NET Core Middleware



Middleware Pipeline



ASP.NET Core Pipeline Order

The order that middleware components are added in the `Startup.Configure` method defines the order in which the middleware components are invoked on requests and the reverse order for the response. The order is critical for security, performance, and functionality.

- Exception/error handling
- HTTP Strict Transport Security Protocol
- HTTPS redirection
- Static file server
- Cookie policy enforcement
- Authentication
- Session
- MVC
- Razor

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseDatabaseErrorPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
        // The default HSTS value is 30 days. You may want to change this for production.
        app.UseHsts();
    }
    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseCookiePolicy();

    app.UseRouting();
    app.UseRequestLocalization();
    app.UseCors();

    app.UseAuthentication();
    app.UseAuthorization();
    app.UseSession();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");
        endpoints.MapRazorPages();
    });
}
```

Demo: Writing Middleware

Module 2: Overview

Section 3: ASP.NET Core

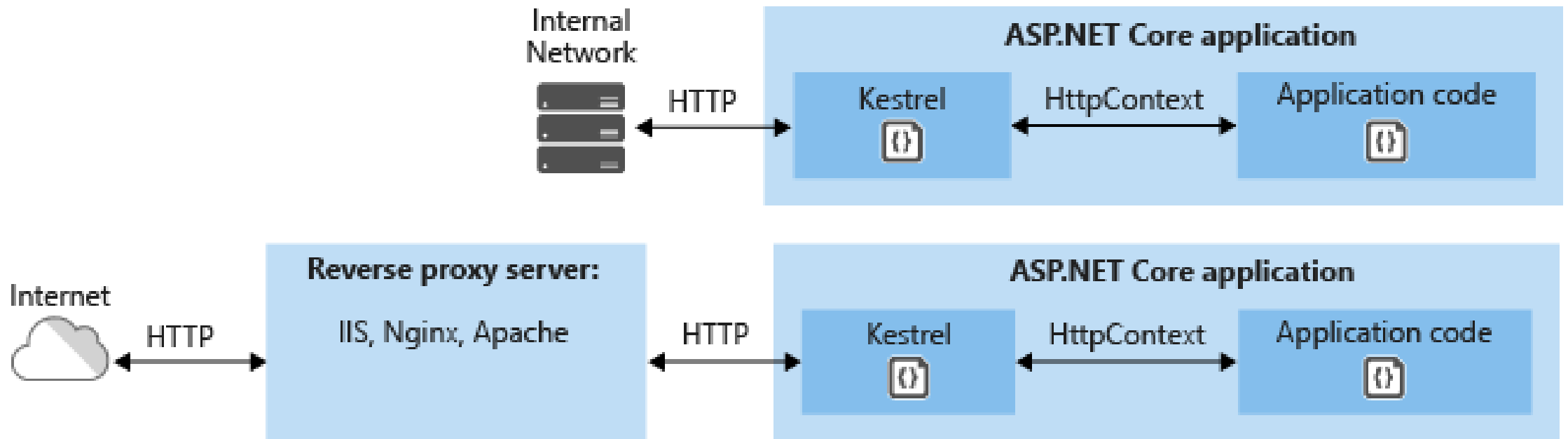
Lesson: Hosting Servers

ASP.NET Core Hosting

- ASP.NET Core is completely decoupled from the web server environment that hosts the application
- ASP.NET Core ships with the following server implementations:
 - [**Kestrel**](#) is the default, cross-platform HTTP server for ASP.NET Core and the default web server specified by the ASP.NET Core project templates.
 - [**HTTP.sys**](#) is only runs on Windows. HTTP.sys is an alternative to [Kestrel](#) server and offers some features that Kestrel doesn't provide. It is formerly called WebListener.

Kestrel

- Kestrel is the default web server included in ASP.NET Core project templates.
- Kestrel can be used by itself or with a reverse proxy server, such as IIS, Nginx, or Apache. A reverse proxy server receives HTTP requests from the Internet and forwards them to Kestrel after some preliminary handling.

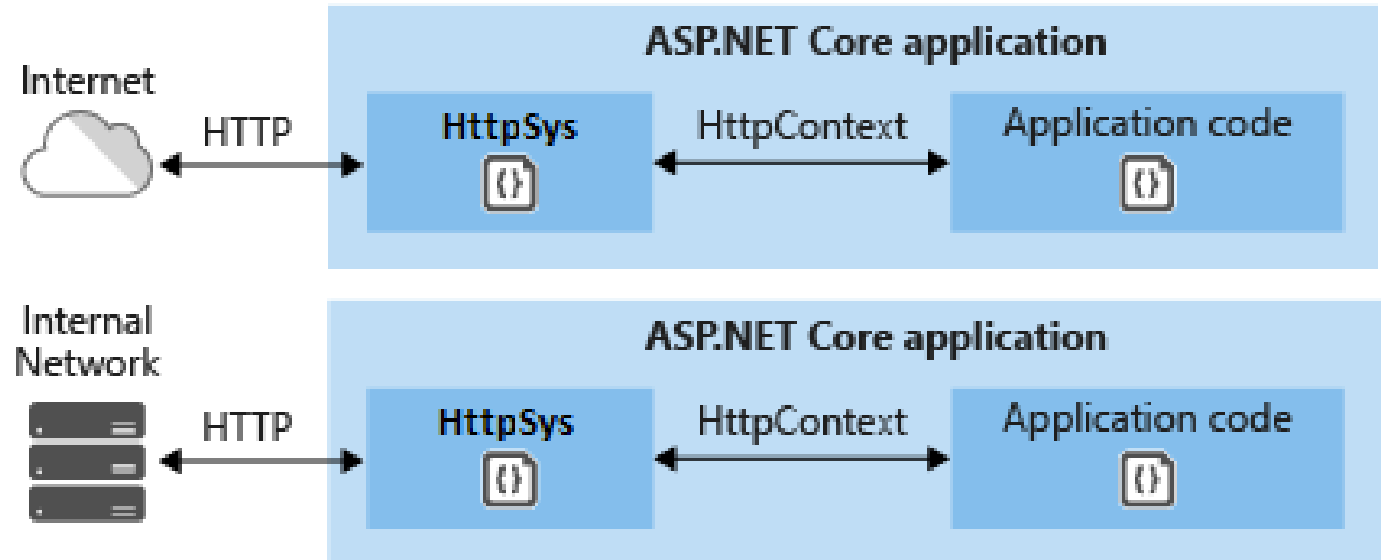


Kestrel Features

- Supported Features
 - HTTPS
 - Opaque upgrade used to enable [WebSockets](#)
 - Unix sockets for high performance behind Nginx
 - HTTP/2 (except on macOS†)
 - Operating system†
 - Windows Server 2016/Windows 10 or later‡
 - Linux with OpenSSL 1.0.2 or later (for example, Ubuntu 16.04 or later)
 - Target framework: .NET Core 2.2 or later
 - [Application-Layer Protocol Negotiation \(ALPN\)](#) connection
 - TLS 1.2 or later connection
- Unsupported Features
 - Sharing an IP address and port across multiple instances without a reverse proxy.
 - SSL when configuring URL bindings using UseUrls

HTTP.sys

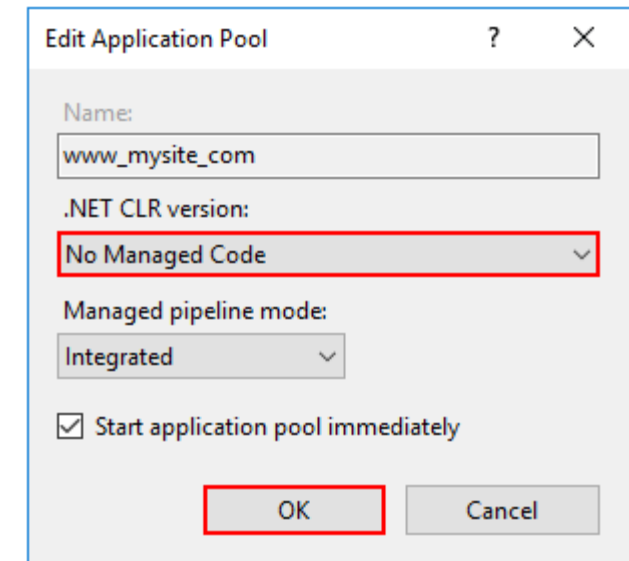
- Supported Features
 - Windows Authentication
 - Port sharing
 - HTTPS with SNI
 - HTTP/2 over TLS (Windows 10)
 - Direct file transmission
 - Response caching
 - WebSockets (Windows 8 or later)



HTTP.sys is incompatible with the ASP.NET Core Module and can't be used with IIS or IIS Express.

Internet Information Services (IIS)

- IIS is used as reverse proxy with Kestrel
- HTTP.sys cannot be used with IIS in a reverse proxy configuration
- To host ASP.NET Core app on IIS you need to install [.NET Core Hosting Bundle](#) (Installed by default when you install .NET Core SDK)
- For a better performance, Application Pool configuration for the website needs to run with No Managed Code
- IIS relies on ANCM (ASP.NET Core Module) to run the apps.



ASP.NET Core Module

The ASP.NET Core Module is a native IIS module that plugs into the IIS pipeline to either:

In-process hosting model

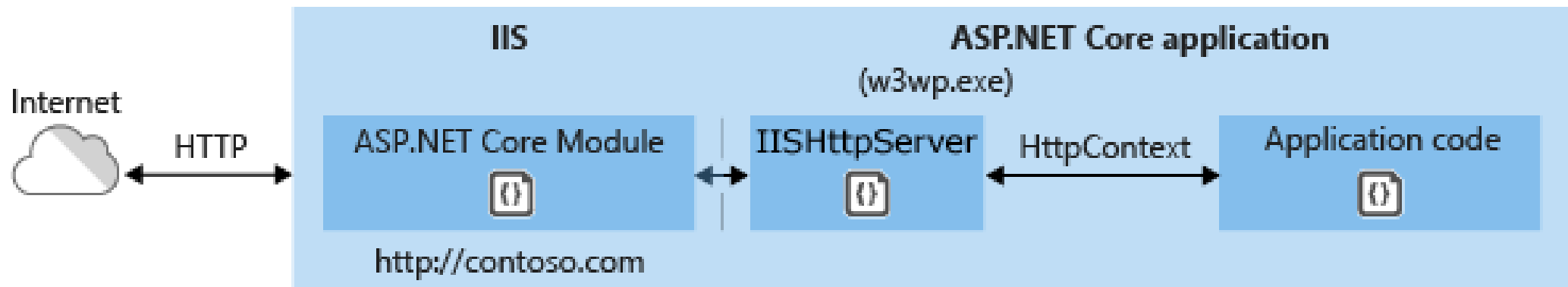
Host an ASP.NET Core app inside of the IIS worker process (w3wp.exe)

Out-of-process hosting model

Forward web requests to a backend ASP.NET Core app running the Kestrel server.

In-process hosting model

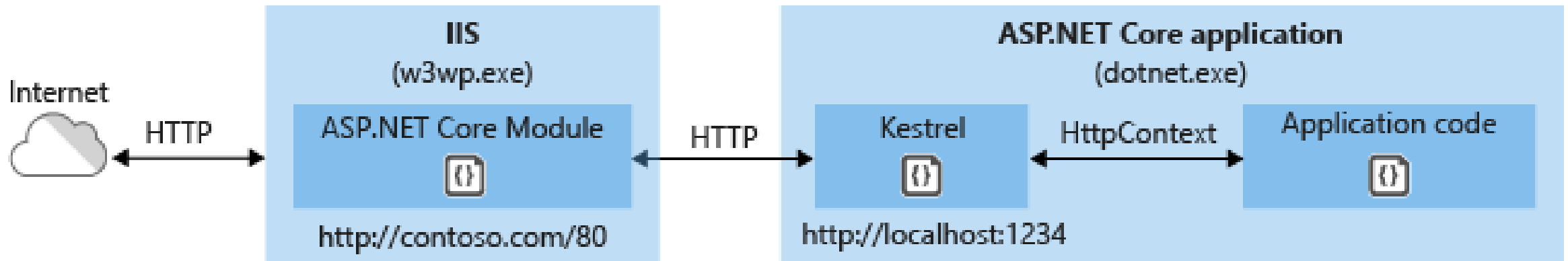
- ASP.NET Core apps default to the in-process hosting model.
- Request flow
 1. Incoming web request is routed to primary port 80/443 through kernel model Http.Sys driver
 2. Request forwarded to ASP.NET Core app (on non-80-443 port)
 3. Kestrel picks up the request and pushes it into ASP.NET Core middleware pipeline
 4. Middleware passes the request to application logic as HttpContext instance
 5. Application HTTP response is eventually passed back to IIS



Out-of-process hosting model

- To configure an app for out-of-process hosting, set the value of the `<AspNetCoreHostingModel>` property to `OutOfProcess` in the project file (.csproj):
- Request flow
 - Loads the CoreCLR.
 - Calls `Program.Main`.

```
AspNetCoreApp1.csproj  + x
1 <Project Sdk="Microsoft.NET.Sdk.Web">
2   <PropertyGroup>
3     <TargetFramework>netcoreapp3.1</TargetFramework>
4     <AspNetCoreHostingModel>OutOfProcess</AspNetCoreHostingModel>
5   </PropertyGroup>
```













Enable ANCM in web.config

- You can create the needed `web.config` file by executing the following command (CLI)
 - `dotnet new webconfig`
- The configuration will enable the ASP.NET Core Module to be executed in the IIS request pipelines

```
web.config  + x
1  <?xml version="1.0" encoding="utf-8"?>
2  <configuration>
3    <location path="." inheritInChildApplications="false">
4      <system.webServer>
5        <handlers>
6          <remove name="aspNetCore" />
7          <add name="aspNetCore"
8            path="*" verb="*"
9            modules="AspNetCoreModuleV2"
10           resourceType="Unspecified" />
11        </handlers>
12        <aspNetCore processPath="%LAUNCHER_PATH%"
13          arguments="%LAUNCHER_ARGS%"
14          stdoutLogEnabled="false"
15          stdoutLogFile=".\logs\stdout" />
16      </system.webServer>
17    </location>
18  </configuration>
```

Which Web Server Should You Use?

Choosing the right Web Server

Web Server	Windows	Linux/OSX	Development-Ready
IIS			
IIS Express			
HTTP.sys			
Kestrel			
Apache/Nginx			

Demo: Kestrel & HTTP.sys hosting

Demo: IIS Hosting

Module 2: Overview

Section 3: ASP.NET Core

Lesson: Working Environments

Working Environments

- Working Environments
 - Development
 - Test
 - Staging
 - Production
- Startup Conventions
 - Startup → Startup{EnvironmentName} – for example, *StartupDevelopment*
 - ConfigureServices() → Configure[Environment]Services()
 - Configure() → Configure[Environment]()
- Applies to Microsoft Azure as well through App Settings in Azure Portal

ASP.NET Core Environments

- ASP.NET Core configures app behavior based on the runtime environment using an environment variable.
- ASP.NET Core reads the environment variable `ASPNETCORE_ENVIRONMENT` at app startup and stores the value in `IWebHostEnvironment.EnvironmentName`
- `ASPNETCORE_ENVIRONMENT` can be set to any value, but three values are provided by the framework:
 - Development
 - Staging
 - Production (default)

Setting Environment Configuration – Visual Studio

The screenshot shows the Visual Studio IDE with the 'web.config' file open. The left sidebar has the 'Debug' tab selected. The main area displays configuration settings for the 'IIS Express' profile. A red rectangle highlights the 'Environment variables' table, which contains one entry: 'ASPNETCORE_ENVIRONMENT' with the value 'Development'.

Configuration: N/A Platform: N/A

Profile: IIS Express New... Delete

Launch: IIS Express

Application arguments: Arguments to be passed to the application

Working directory: Absolute path to working directory Browse...

☒ Launch browser: Absolute or relative URL

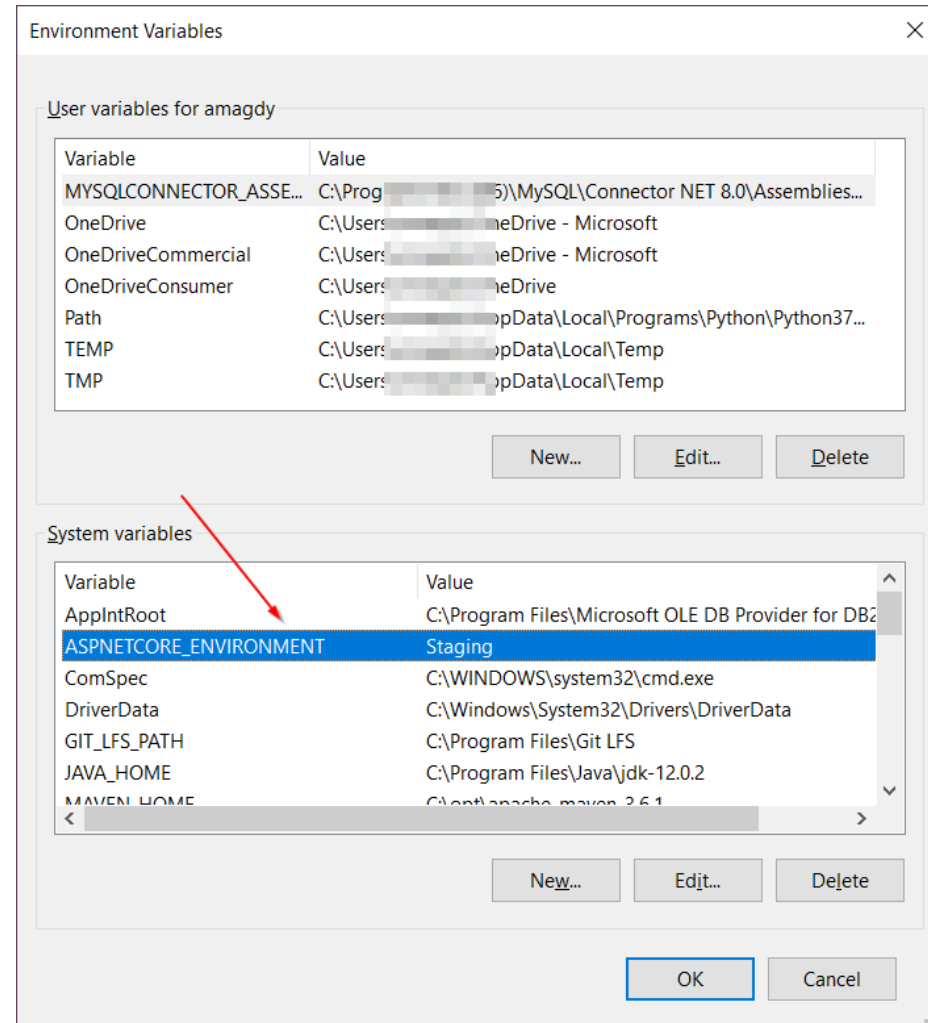
Environment variables:

Name	Value
ASPNETCORE_ENVIRONMENT	Development

Add Remove

☐ Enable native code debugging

Setting Environment Configuration – Windows



Demo: Working Environments

Module Summary

- In this module, you learned the following:
 - What is ASP.NET Core
 - One ASP.NET
 - ASP.NET Core Projects
 - ASP.NET Core Configurations
 - ASP.NET Core Middleware
 - ASP.NET Core Hosting Servers
 - Working Environments



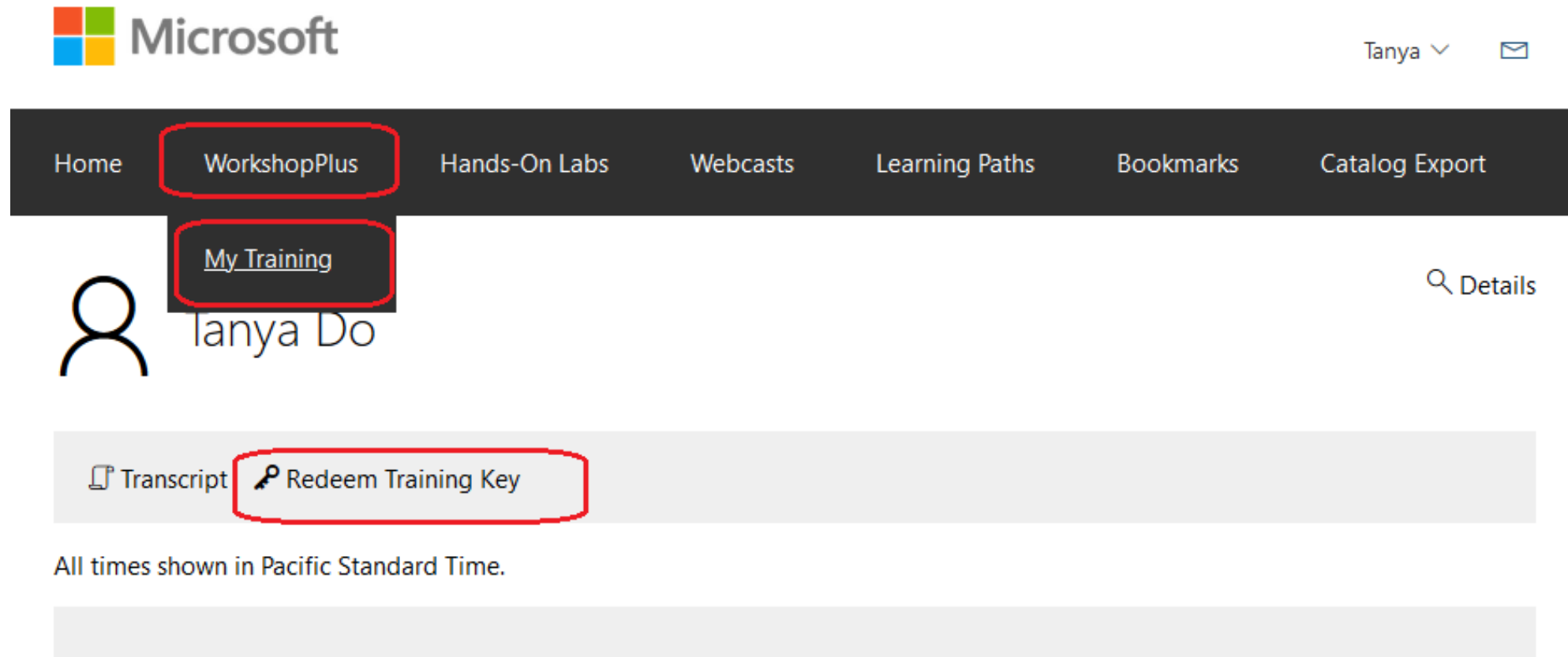
Lab: Explore Visual Studio Tooling, ASP.NET Core



Connect to Lab environment LOD

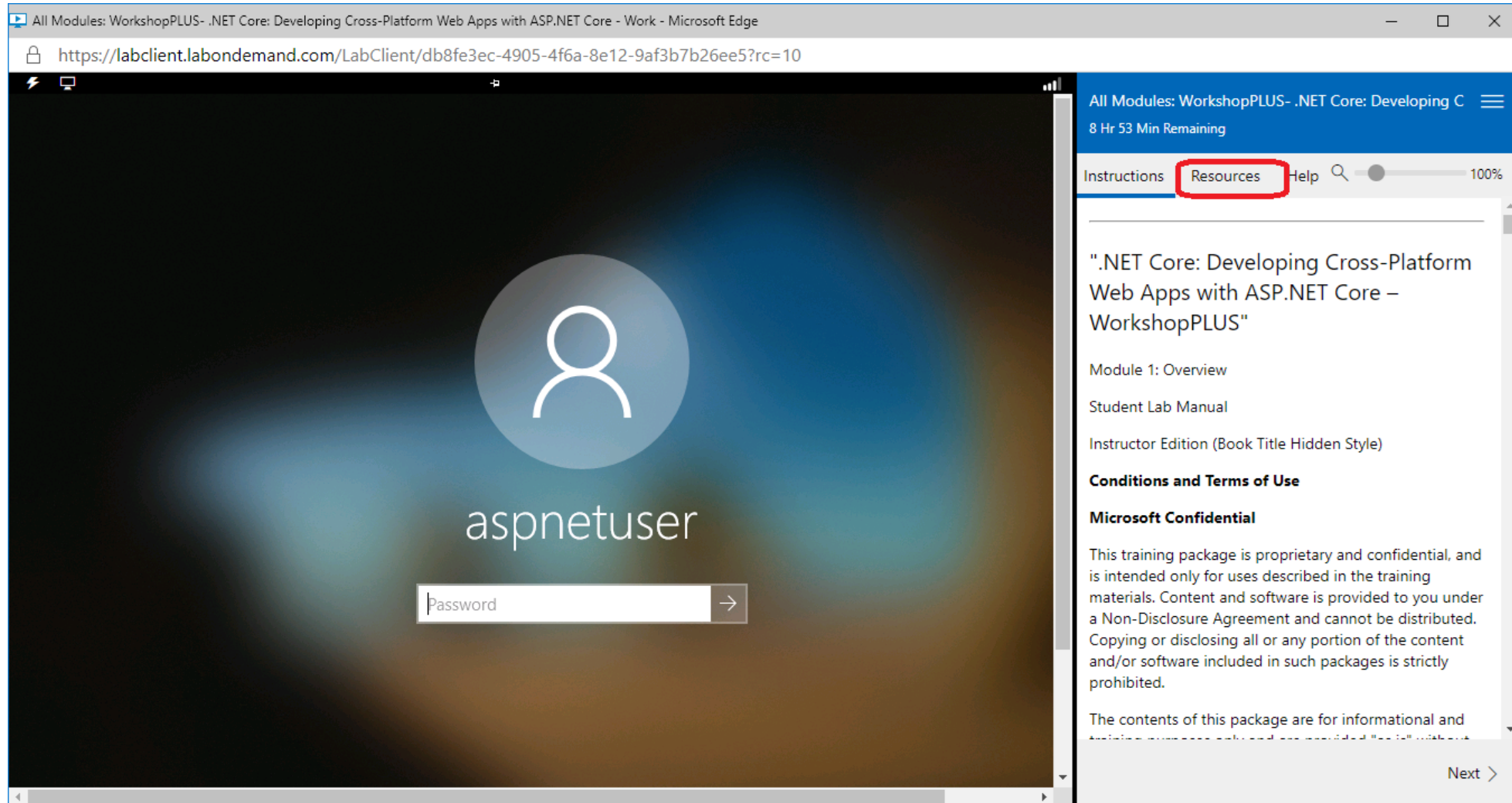
- Using your browser go to any of the following links:
 - <https://aka.ms/premiereducation>
 - <https://aka.ms/lod>
- Sign in with your Microsoft account (Live ID/Azure Id)
 - *If you do not have one, then create it here: <http://live.com>*
- Click "Redeem Training Key"
- Register the lab code **YOUR LAB Code**

How to “redeem” the training key



The screenshot displays the Microsoft Learning portal interface. At the top left is the Microsoft logo. In the top right corner, the user's name 'Tanya' is shown with a dropdown arrow and an envelope icon. A dark navigation bar contains the following links: 'Home', 'WorkshopPlus' (highlighted with a red box), 'Hands-On Labs', 'Webcasts', 'Learning Paths', 'Bookmarks', and 'Catalog Export'. Below the navigation bar, on the left, is a user profile section with a person icon and the name 'Tanya Do'. A dropdown menu is open under the profile, showing 'My Training' (highlighted with a red box). To the right of the profile is a 'Details' link with a magnifying glass icon. Below the navigation bar, there is a light gray bar containing two options: 'Transcript' with a document icon and 'Redeem Training Key' with a key icon (highlighted with a red box). Below this bar, a note states 'All times shown in Pacific Standard Time.' followed by a large empty gray rectangular area.

"Resource" tab for login password



The screenshot shows a web browser window with the address bar displaying `https://labclient.labondemand.com/LabClient/db8fe3ec-4905-4f6a-8e12-9af3b7b26ee5?rc=10`. The main content area features a login screen with a large circular icon containing a white person silhouette, the text "aspnetuser", and a password input field labeled "Password" with a right-pointing arrow button. The sidebar on the right has a blue header with the text "All Modules: WorkshopPLUS- .NET Core: Developing C" and "8 Hr 53 Min Remaining". Below the header, there are three tabs: "Instructions", "Resources" (highlighted with a red rectangle), and "Help". The "Resources" tab is active, displaying the title ".NET Core: Developing Cross-Platform Web Apps with ASP.NET Core – WorkshopPLUS" and a list of items: "Module 1: Overview", "Student Lab Manual", and "Instructor Edition (Book Title Hidden Style)". Below these items, there is a section titled "Conditions and Terms of Use" with the heading "Microsoft Confidential". The text in this section states: "This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited." At the bottom of the sidebar, there is a "Next >" button.

