多奇·教育訓練

ASP.NET Core 5 開發實戰

部署維運篇

多奇數位創意有限公司

技術總監 黃保翕（Will 保哥）

https://blog.miniasp.com

Configurations

ASP.NET Core 組態管理

ASP.NET Core 組態設定的大致用法

- 設定**組態提供者** (Configuration Providers)
  - ASP.NET Core 內建已有 JSON file, User Secrets, Environment Variables, Command Line Arguments 等組態提供者！
- 在 **Startup** 類別的 **ConfigureServices()** 設定 DI 容器
  - 預設專案範本會將 IConfiguration 注入到 Startup 類別中
  - 你可以直接將其加入 DI 容器中，或是轉強型別後再加入到 DI 容器中
- 在**控制器**或**其他服務類別**中注入並**取得設定**

關於 ASP.NET Core 的組態設定

- **組態設定**皆由 **設定提供者** (Configuration Providers) 負責
  - 從**文字檔案**取得組態設定 - 支援 `INI`、`JSON` 與 `XML`
  - 從**命令列參數**取得組態設定
  - 從**環境變數**取得組態設定
  - 從**記憶體中的 .NET 物件**取得組態設定
  - 從**未加密的 Secret Manager 儲存區**取得設定
  - 從**有加密的使用者儲存區** (如 Azure Key Vault) 取得設定
  - 也可**自訂組態提供者** (例如透過 Entity Framework 取得組態設定)
- **組態設定**最終都會變成 **key-value pairs** 的形式
  - 所有 Key 與 Value 一定是「**字串**」型態
  - 組態設定很容易可被**反序列化**為任何 POCO 物件 (有內建 API 處理)
  - 組態設定使用 IConfiguration 型別取得設定值

ASP.NET Core 的選項模式

- 繫結**頂層**設定到 .NET 物件並加入服務集合

```
- services.Configure<AppSettings>(Configuration);
```

- 繫結**子層**設定到 .NET 物件並加入服務集合

```
- services.Configure<AppSettings>(Configuration.GetSection("App"));
```

- 繫結**子子層**設定到 .NET 物件並加入服務集合

```
- services.Configure<AppSettings>(Configuration.GetSection("App:Module1"));
```

- **注意事項**
  - 這裡的 Configuration 物件，其組態設定來源不只一個！
  - 請參閱 ASP.NET Core 中的選項模式

注入 IOptions<T> 選項物件

- IOptions<AppSettings>
  - 直接取得 AppSettings 的選項物件
- IOptionsSnapshot<AppSetting>
  - 這種寫法會監視 **appsettings.json** 檔案變更 (不用重啟應用程式)
- 語法範例

```
public AppSettings Settings { get; }
public ValuesController(IOptions<AppSettings> options)
{
    Settings = options.Value;
}
```

ASP.NET Core 的設定繫結

- 繫結頂層設定到 .NET 物件
  - `var appSettings = new AppSettings(); // 先定義 AppSettings 類別`
  - `Configuration.Bind(appSettings);`

    或以下語法
  - `var appSettings = Configuration.Get<AppSettings>();`
- 繫結子層設定到 .NET 物件
  -
```
var appSettings =
    Configuration.GetSection("App").Get<AppSettings>();
```
- 繫結子子層設定到 .NET 物件
  -
```
var appSettings =
    Configuration.GetSection("App:Module1").Get<AppSettings>();
```

注入 AppSettings 自定義物件

- 將強型別的設定物件加入為 Singleton 服務物件
  - `services.AddSingleton(appSettings);`
- 服務注入方法

```
public ValuesController(AppSettings appSettings)
{
}
```

- 不用特別將 IConfiguration 物件加入 DI 容器也可注入組態物件
  - ~~`services.AddSingleton(Configuration);`~~

```
public ValuesController(IConfiguration config)
{
}
```

在 ASP.NET Core MVC 的 View 注入服務

- 在 ASP.NET MVC 的 View 中注入 **IConfiguration** 物件

```
@inject IConfiguration Configuration
```

- 在 ASP.NET MVC 的 View 中注入 **IOptions<T>** 物件

```
@inject IOptions<AppSetting> OptionsAccessor
```

```
@inject IOptionsSnapshot<AppSetting> OptionsAccessor
```

預設取得資料庫連接字串的 APIs

- 設定檔 (組態) 的結構定義 (appsettings.json)

```
{
    "ConnectionStrings": {
        "DefaultConnection": "Server=(localdb)\\MSSQLLocalDB; ……"
    }
}
```

- 取得連接字串

```
Configuration.GetConnectionString("DefaultConnection")

Configuration["ConnectionStrings:DefaultConnection"]
```

設定提供者的套用順序 (依序套用設定)

- AddJsonFile("appsettings.json",
  optional : true, reloadOnChange : true)
- AddJsonFile($"appsettings.{env.EnvironmentName}.json",
  optional : true, reloadOnChange : true)
- AddUserSecrets(appAssembly, optional : true)
  - 這個設定預設只有在「開發環境」(**Development**) 才會套用
  - 透過 **dotnet publish** 發行的網站預設為 **Production** 環境
- AddEnvironmentVariables()
- AddCommandLine(args) // 優先權最高 (會覆寫先前設定)

初始化 使用者安全變數 設定

- 手動設定 **UserSecretsId** 屬性 (MSBuild)

mvc1.csproj

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <Project Sdk="Microsoft.NET.Sdk.Web">
3     <PropertyGroup>
4       <TargetFramework>netcoreapp3.1</TargetFramework>
5       <UserSecretsId>3b564c43-bd26-4f2a-b646-f9e214e51f2a</UserSecretsId>
6     </PropertyGroup>
7   </Project>
```

- .NET Core 3 支援透過 .NET Core CLI 初始化設定 (自動修改專案檔)
  - **dotnet user-secrets init**
- 實體檔案所在路徑
  - Windows: **%APPDATA%\Microsoft\UserSecrets\<user_secrets_id>\secrets.json**
  - Linux/macOS: **~/.microsoft/usersecrets/<user_secrets_id>/secrets.json**
- **注意事項**：所有設定值皆為**明碼**儲存！

透過 .NET Core CLI 管理使用者安全變數

- 列出目前專案所設定的安全組態設定清單
  - `dotnet user-secrets list`
  - `dotnet user-secrets list --id UserSecretsId`
- 設定使用者安全變數
  - `dotnet user-secrets set KEY VALUE`
  - `dotnet user-secrets set KEY VALUE --id UserSecretsId`
  - `type .\input.json | dotnet user-secrets set`
  - **注意**：**多階層組態**必須用 **冒號**（**:**）間隔。例如：
    `dotnet user-secrets set "ConnectionStrings:DefaultConnection" "……"`
- 移除使用者安全變數
  - `dotnet user-secrets remove KEY`
- 清空使用者安全變數
  - `dotnet user-secrets clear`

從環境變數取得組態設定

- 不同區段之間用**兩個底線**分隔！

```
App__Module1__OK=12345

ConnectionStrings__DefaultConnection=……

Logging__LogLevel__Default=Warning
```

- **透過容器執行 .NET Core 時，經常利用環境變數設定參數！**

從命令列參數取得組態設定

- 四種不同的參數設定格式皆可使用

```
dotnet run -- --key1=value1 --key2=value2
dotnet run -- -key1=value1 -key2=value2
dotnet run -- /key1=value1 /key2=value2
dotnet run -- key1=value1 key2=value2
```

- 也可以混合不同的參數格式

```
dotnet run -- key1=value1 -key2=value2 --key3=value3
```

- 也可以使用冒號(:)分隔組態的區段

```
dotnet run -- --App:Module1:OK=123
```

Logging

ASP.NET Core 紀錄管理

內建的紀錄提供者

- Console (預設已加入)
  - `logging.AddConsole();`
- Debug (預設已加入)
  - `logging.AddDebug();`
- EventSource
  - `logging.AddEventSourceLogger();`
- EventLog
  - `logging.AddEventLog();`
- TraceSource
  - `logging.AddTraceSource(sourceSwitchName);`
- Azure App Service
  - `logging.AddAzureWebAppDiagnostics();`

調整 CreateDefaultBuilder() 預設提供者

- Program.cs

```
public static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .ConfigureWebHostDefaults(webBuilder =>
        {
            webBuilder.UseStartup<Startup>()
                .ConfigureLogging(logging => {
                    logging.ClearProviders();
                    logging.AddConsole();
                });
        });
```

取得 ILogger 物件

- 在 Startup 或 Controller 注入 **ILogger** 的方法

```
private readonly ILogger _logger;
public HomeController(ILogger<HomeController> logger)
{
    this._logger = logger;
}
```

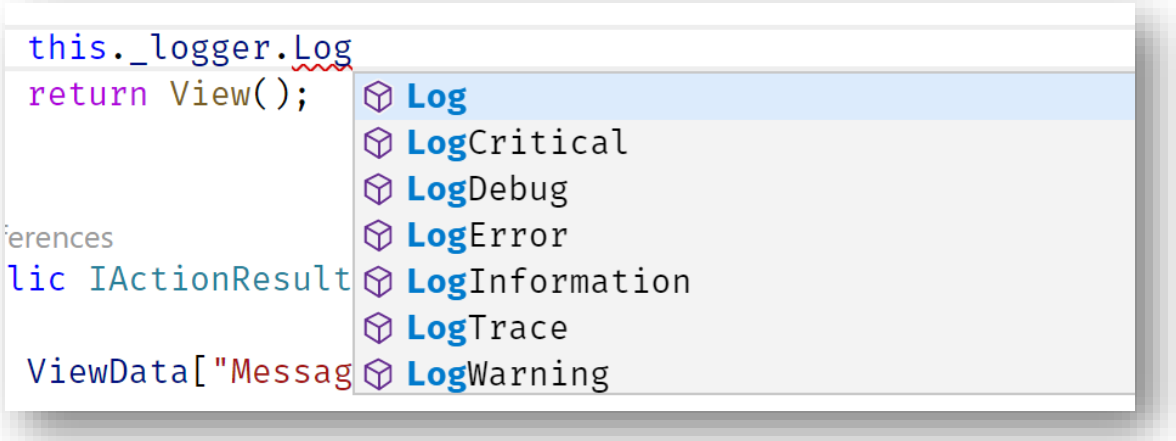
預設會建立記錄類別
(Log category)

- 在 Program 類別取得 **ILogger** 的方法

```
var host = CreateWebHostBuilder(args).Build();
var logger = host.Services.GetRequiredService<ILogger<Program>>();
logger.LogInformation("My Log Here!");
host.Run();
```

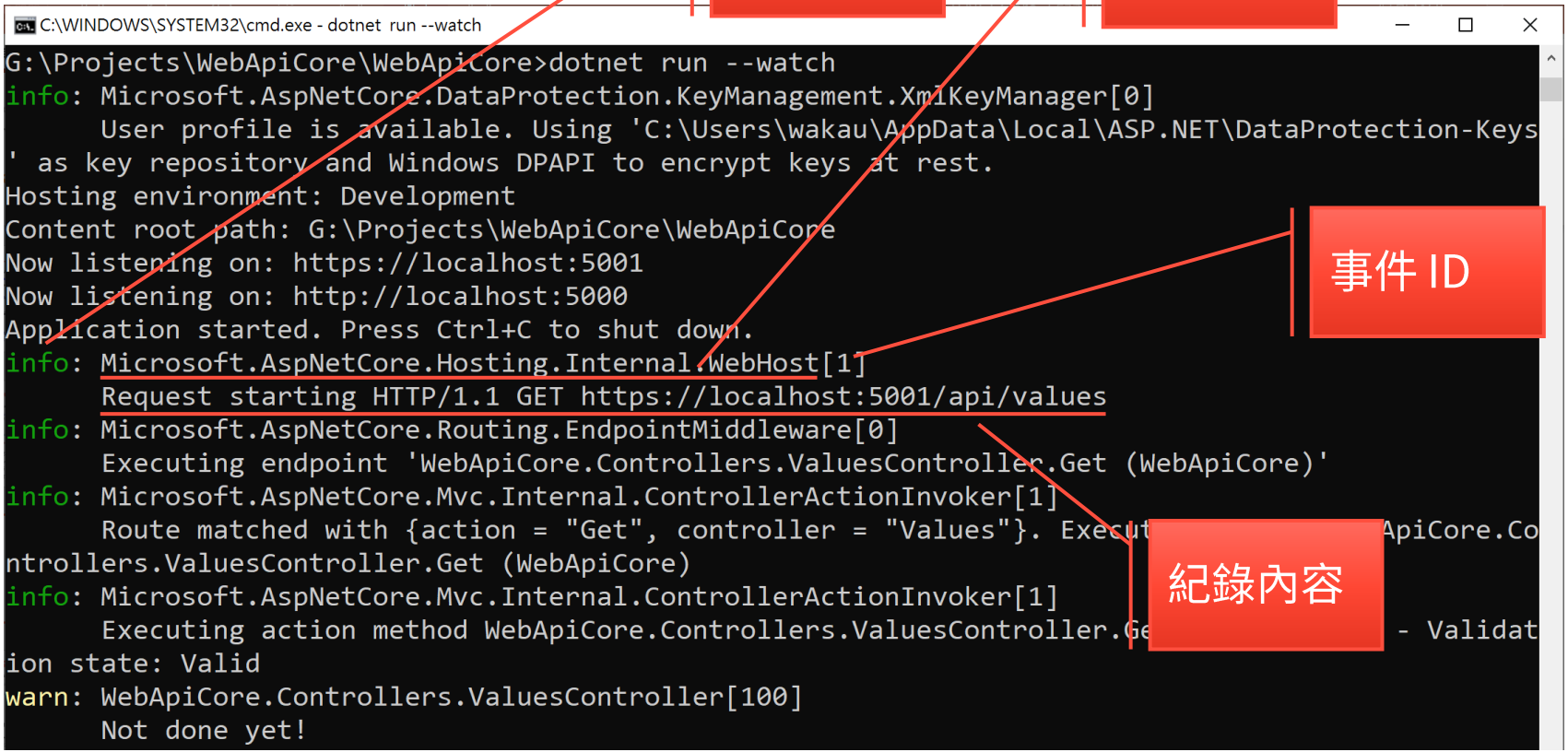
使用 ILogger 的方法

- 常用的 Log 用法



- 沒有非同步的 APIs 可用
  - 所有的 Log APIs 都是「同步式」的
  - 這意味著所有 Log 動作都應該越快完成越好
  - 如果要寫入 DB 建議先寫入 **Queue** 再透過**背景工作**寫入資料庫

常見記錄格式



設定記錄詳細等級

- 預設會從 appsettings.json 的 "Logging" 區段進行設定

appsettings.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Warning"
    }
  }
}
```

- **紀錄等級**(Logging:LogLevel)用來設定不同**紀錄類別**下的紀錄詳細程度
  - Trace, Debug, Information, Warning, Error, Critical, None
- 這裡的 **"Default"** 是一個「**預設紀錄類別**」
  - 用來設定所有不在 **"LogLevel"** 清單中定義的**預設紀錄等級**
  - 沒有設定**紀錄篩選條件**的時候, 就會以 **"Default"** 等級套用

開發環境下的紀錄詳細等級

- appsettings.Development.json

appsettings.Development.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Debug",
      "System": "Information",
      "Microsoft": "Information"
    }
  }
}
```

- **預設**所有**紀錄等級**只到 **Debug**
- 設定所有 **System.*** 紀錄類別下的**紀錄等級**皆設定為 **Information**
- 設定所有 **Microsoft.*** 紀錄類別下的**紀錄等級**皆設定為 **Information**

關於記錄類別 (Log category)

- 透過 `ILogger<HomeController>` 注入的物件
  - 預設的**紀錄類別**是該型別的**完整名稱** (Full Name) (包含**命名空間**部分)
    例如： **`api1.Controllers.HomeController`**
- 透過 `ILoggerFactory` 注入的物件
  - 你可以透過 CreateLogger 來建立**自定的紀錄類別**
  -
```
ILogger logger =
logger.CreateLogger("api1.Controllers.HomeController");
```
- 你可透過 **紀錄篩選** 機制，過濾掉特定**記錄類別**的內容

紀錄篩選的設定範例

```
{
  "Logging": {
    "Debug": {
      "LogLevel": {
        "Default": "Information"
      }
    },
    "Console": {
      "IncludeScopes": false,
      "LogLevel": {
        "Microsoft.AspNetCore.Mvc.Razor.Internal": "Warning",
        "Microsoft.AspNetCore.Mvc.Razor.Razor": "Debug",
        "Microsoft.AspNetCore.Mvc.Razor": "Error",
        "Default": "Information"
      }
    },
    "LogLevel": {
      "Default": "Debug"
    }
  }
}
```

針對特定紀錄提供者 (Debug)

僅針對 Console 紀錄提供者做設定

預設紀錄等級

紀錄篩選的提供者別名 (Provider aliases)

- 以下是你可以用在組態設定時的提供者別名
  - Console
  - Debug
  - EventSource
  - EventLog
  - TraceSource
  - AzureAppServicesFile
  - AzureAppServicesBlob
  - ApplicationInsights

正確的設定 Log message template

- 善用結構化的紀錄訊息
  - semantic logging, also known as structured logging.
  - `_logger.LogInformation("Getting item {ID} at {RequestTime}", id, DateTime.Now);`
  - 某些第三方的紀錄提供者，有提供「結構化」的紀錄方式，只要能按照上述的方式進行記錄，就可以依據命名與欄位儲存到可被檢索的訊息格式。例如： Azure Table Storage、SQL Server、ElasticSearch

- **千萬不要用「組字串」的方式撰寫紀錄訊息！**

記錄範圍 (Log scopes)

- 群組相關的紀錄訊息

```
public IActionResult GetById(string id)
{
    TodoItem item;
    using (_logger.BeginScope("Message attached to logs created in the using block"))
    {
        _logger.LogInformation(LoggingEvents.GetItem, "Getting item {ID}", id);
        item = _todoRepository.Find(id);
        if (item == null)
        {
            _logger.LogWarning(LoggingEvents.GetItemNotFound, "GetById({ID}) NOT FOUND", id);
            return NotFound();
        }
    }
    return new ObjectResult(item);
}
```

設定顯示記錄範圍

- 設定輸出的紀錄必須包含**範圍資訊**
- ASP.NET Core 預設會包含 `ConnectionId` 與 `RequestId` 範圍

appsettings.Development.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Debug",
      "System": "Information",
      "Microsoft": "Information"
    },
    "Console": {
      "IncludeScopes": "true"
    }
  }
}
```

好用的第三方紀錄提供者

- Serilog (GitHub repo) (Serilog.AspNetCore) (Provided Sinks)
- elmah.io (GitHub repo)
- Gelf (GitHub repo)
- JSNLog (GitHub repo)
- KissLog.net (GitHub repo)
- Loggr (GitHub repo)
- NLog (GitHub repo)
- Sentry (GitHub repo)
- Stackdriver (Github repo)

Serilog

- Serilog | Serilog.AspNetCore | Configuration | Provided Sinks
- `dotnet add package Serilog.AspNetCore`
- 設定三部曲
  1. 初始化 Serilog 紀錄器
  2. 設定 CreateHostBuilder 使用 Serilog
  3. 移除 appsettings.json 中的 **"Logging"** 區段
- 啟用較為精簡的要求紀錄 (Request logging)
  - 加入 **Microsoft.AspNetCore** 紀錄等級到 **LogEventLevel.Warning**
  - 加入 **app.UseSerilogRequestLogging();** 到**要求管線**中 (Configure)

初始化 Serilog 紀錄器

```
Log.Logger = new LoggerConfiguration()
    .MinimumLevel.Debug()
    .MinimumLevel.Override("Microsoft", LogEventLevel.Information)
    .Enrich.FromLogContext()
    .WriteTo.Console()
    .CreateLogger();

try
{
    CreateHostBuilder(args).Build().Run();
}
finally
{
    Log.CloseAndFlush();
}
```

設定 CreateHostBuilder 使用 Serilog

```
public static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .ConfigureWebHostDefaults(webBuilder =>
        {
            webBuilder.UseStartup<Startup>();
        })
        .UseSerilog();
```

Seq — centralized structured logs

- https://datalust.co/seq
- 透過 Docker 執行 (datalust/seq)
  - `docker pull datalust/seq:latest`
  - `docker run -e ACCEPT_EULA=Y -p 5341:80 datalust/seq:latest`
  - http://localhost:5341/
- 加入 Serilog.Sinks.Seq 套件
  - dotnet add package Serilog.Sinks.Seq
  - 加入 Log.Logger 設定
    - `.WriteTo.Seq("http://localhost:5341")`

Use multiple environments in ASP.NET Core

ASP.NET Core 多環境設定

認識「多環境」設定

- ASP.NET Core 會在應用程式啟動時
  - 會自動讀取 **ASPNETCORE_ENVIRONMENT** 環境變數！
  - 環境名稱可自訂，若未設定該環境變數，預設值為 **Production**
  - 該環境變數可以透過 IWebHostEnvironment 取得
    - 記得要引用 **Microsoft.Extensions.Hosting** 命名空間 (有擴充方法)
- 內建支援三種環境 (以下三個都是回傳**字串**類型)
  - EnvironmentName.Development
  - EnvironmentName.Production
  - EnvironmentName.Staging
- 透過開發工具自訂環境名稱
  - **Properties\launchSettings.json** (VS2017) (.NET CLI)
  - **.vscode/launch.json** (VSCode)

執行 dotnet run 時如何設定環境變數

- Command Prompt
  - `set ASPNETCORE_ENVIRONMENT=Development`
  - `setx ASPNETCORE_ENVIRONMENT=Development /M`
- PowerShell
  - `$Env:ASPNETCORE_ENVIRONMENT = "Development"`
  - `[Environment]::SetEnvironmentVariable("ASPNETCORE_ENVIRONMENT", "Development", "Machine")`
- Bash (Linux / macOS)
  - `ASPNETCORE_ENVIRONMENT="Development" dotnet run`
  - `export ASPNETCORE_ENVIRONMENT=Development`

執行在 IIS 時如何設定環境變數

- 設定環境變數（如果未設定預設就是 **Production**）

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.webServer>
    <handlers>
      <add name="aspNetCore" path="*" verb="*"
           modules="AspNetCoreModule" resourceType="Unspecified" />
    </handlers>
    <aspNetCore processPath="dotnet" arguments=".\m1.dll"
                stdoutLogEnabled="false" stdoutLogFile=".\logs\stdout">
      <environmentVariables>
        <environmentVariable name="ASPNETCORE_ENVIRONMENT" value="Development" />
      </environmentVariables>
    </aspNetCore>
  </system.webServer>
</configuration>
```

環境名稱的使用方式

- Startup.cs
  - 引用 **Microsoft.Extensions.Hosting** 命名空間
  - 注入 **IWebHostEnvironment env** 服務物件
    - `env.IsDevelopment()`
    - `env.IsStaging()`
    - `env.IsProduction()`
    - `env.IsEnvironment("YourName")`
    - `env.EnvironmentName`
- ASP.NET Core 的 <environment> TagHelper

```
<environment names="Staging,Production">
  <strong>Staging or Production</strong>
</environment>
```

- 在 ASP.NET Core 中使用多個環境

多環境下的 Startup 啟動類別 (1)

- 啟動類別命名規則
  - Startup{環境名稱}　　優先選擇有環境名稱的類別
    - StartupDevelopment
    - StartupProduction
    - StartupStaging
  - Startup　　最終預設選取的啟動類別名稱
- 需調整 **CreateWebHostBuilder()** 的寫法

```
public static IWebHostBuilder CreateWebHostBuilder(string[] args)
{
  var assemblyName = typeof(Startup).GetTypeInfo().Assembly.FullName;
  return WebHost.CreateDefaultBuilder(args) .UseStartup(assemblyName);
}
```

多環境下的 Startup 啟動類別 (2)

- 在同一個 Startup 類別下的方法命名規則

  - ConfigureDevelopmentServices()
  - ConfigureProductionServices()
  - ConfigureStagingServices()

  - ConfigureDevelopment()
  - ConfigureProduction()
  - ConfigureStaging()
  - Configure()

部署 ASP.NET Core 應用程式

在正式機啟動 ASP.NET Core 網站的方式

- **Windows 命令提示字元 (注意: 請不要加上雙引號)**

```
SET ASPNETCORE_URLS=http://*:5100 && dotnet mysite.dll
```

- **Windows PowerShell**

```
$env:ASPNETCORE_URLS="http://*:5100" ; dotnet mysite.dll
```

- **Linux / macOS**

```
ASPNETCORE_URLS="http://*:5100" dotnet mysite.dll
```

部署 ASP.NET Core 網站到 Linux

- 建立發行檔案
  - **`dotnet publish -c Release`**
- 部署到 Nginx 網站伺服器
  - Host ASP.NET Core on Linux with Nginx
- 部署到 Apache 網站伺服器
  - Host ASP.NET Core on Linux with Apache

部署 ASP.NET Core 網站到 Docker

- 準備 Dockerfile 檔案

```
FROM microsoft/dotnet:2.1-aspnetcore-runtime
WORKDIR /app
COPY bin/Debug/netcoreapp2.1/publish/. .
ENTRYPOINT ["dotnet", "mymvc.dll"]
```

- 建置容器映像

```
docker build -t mymvc .
```

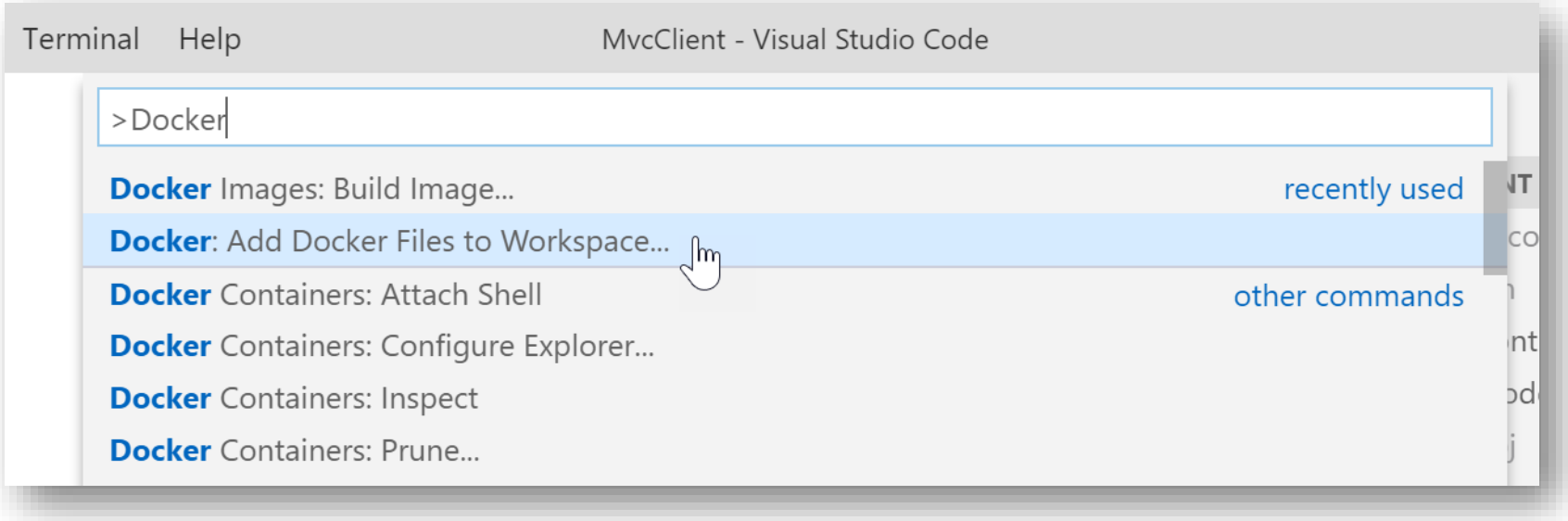
- 執行容器

```
docker run --rm --name=mymvc -d -p 80:80 mymvc
docker logs mymvc
```

- 參考文件
  - 在 Docker 容器中裝載 ASP.NET Core
  - 如何將 ASP.NET Core 2.1 網站部署到 Docker 容器中

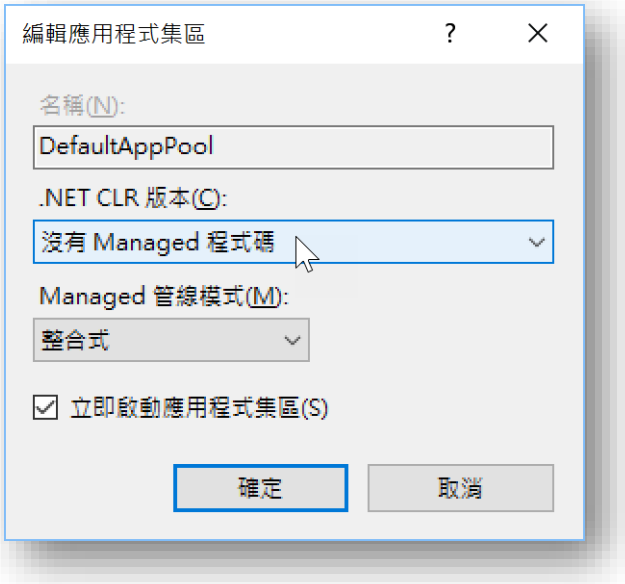
使用 VSCode Docker 擴充套件

- F1 > 搜尋 Docker > **Docker: Add Docker Files to Workspace**
- 該擴充套件會自動分析專案內容，自動產生必要的 Dockerfile



安裝 ASP.NET Core 模組到 IIS 伺服器

- 先安裝 IIS 角色服務並啟用相關功能
- 下載並安裝 ASP.NET Core Runtime Hosting Bundle
- 重新啟動 WAS 服務
  - **net stop was /y**
  - **net start w3svc**
- 新增 IIS 站台
- 設定應用程式集區
  - 調整 **.NET CLR 版本** 設定 (如右圖)
  - 設定為「**沒有 Managed 程式碼**」



部署 ASP.NET Core 網站到 IIS

- 透過 .NET CLI 發行網站
  - `dotnet publish -c Release`
  - `dotnet publish -c Release /p:PublishDir=c:/WebRoot/MyCoreWebsite`
- 透過 Visual Studio 發行網站
  - 適用於 ASP.NET Core 應用程式部署的 Visual Studio 發行設定檔
  - 從命令列發佈至 MSDeploy 端點
- 詳細步驟參考：Host ASP.NET Core on Windows with IIS
- **第二次部署**須設定 ***.csproj** 專案屬性 (不要產生 web.config 檔案)

```
<PropertyGroup>
    <IsTransformWebConfigDisabled>true</IsTransformWebConfigDisabled>
</PropertyGroup>
```

聯絡資訊

The Will Will Web

網路世界的學習心得與技術分享

http://blog.miniasp.com/

Facebook

Will 保哥的技術交流中心

http://www.facebook.com/will.fans

Twitter

https://twitter.com/Will_Huang

多奇·教育訓練

THANK YOU!

Q&A