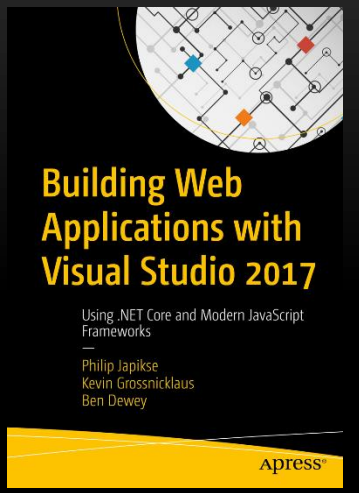
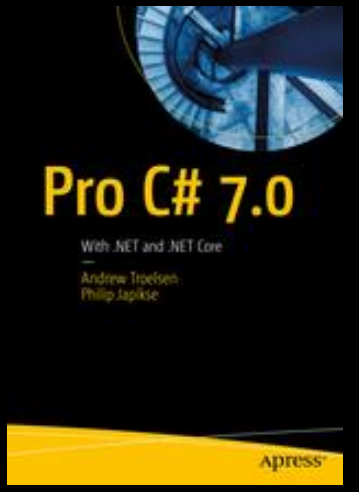
BUILD AN ASP.NET CORE 2.0 AND EF CORE 2.0 APP HANDS ON LAB

Philip Japikse (@skimedic)
skimedic@outlook.com
www.skimedic.com/blog
Microsoft MVP, ASPInsider, MCSD, MCDBA, CSM, CSP
Consultant, Teacher, Writer

Phil.About()

- Consultant, Coach, Author, Teacher
  - Lynda.com (http://bit.ly/skimediclyndacourses)
  - Apress.com (http://bit.ly/apressbooks)
- Microsoft MVP, ASPInsider, MCSD, MCDBA, CSM, CSP
- Founder, Agile Conferences, Inc.
  - http://www.dayofagile.org
- President, Cincinnati .NET User's Group





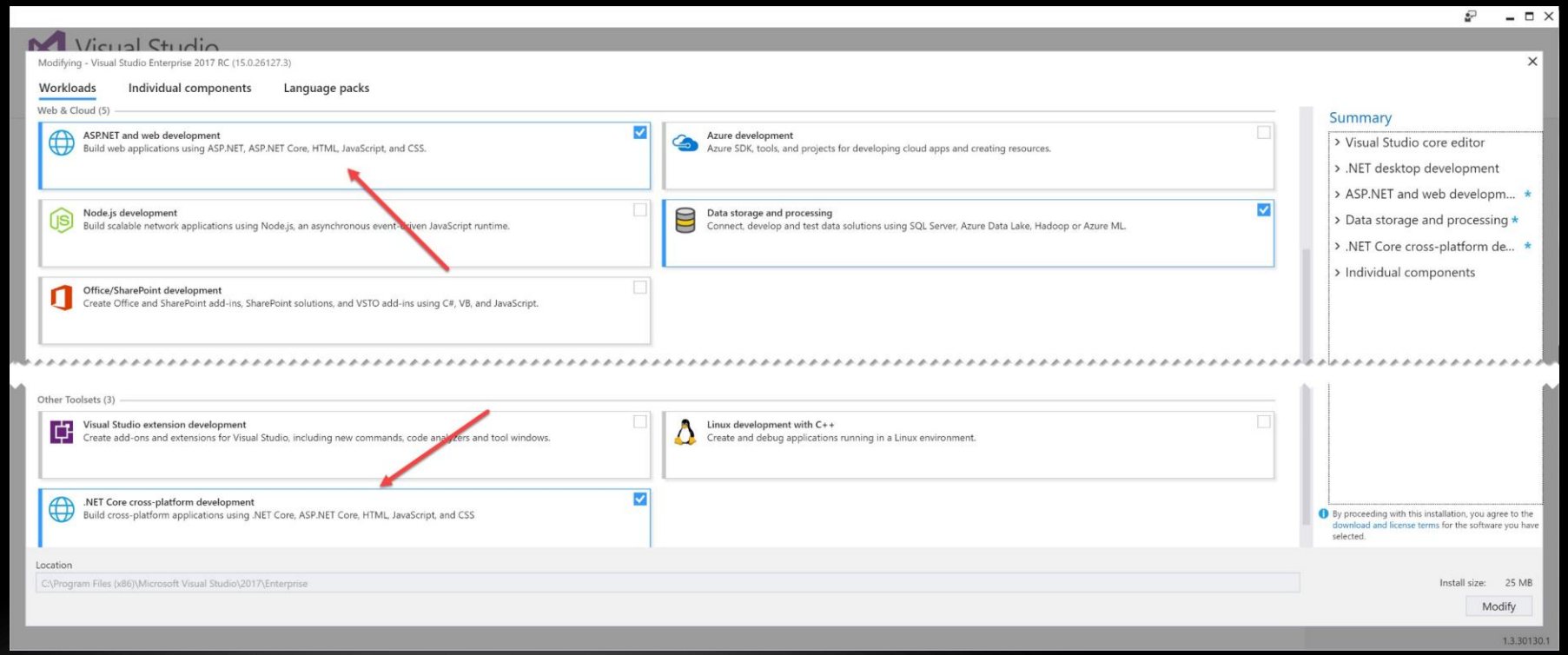
PREREQUISITES

- Visual Studio 2017 15.3 (any edition)
- .NET Core 2.0 SDK
- SQL Server 2016 (any edition)
- 2.0 Lab files from GitHub repo:
  - https://github.com/skimedic/dotnetcore_hol

INSTALLING VS2017 AND .NET CORE

INSTALL VISUAL STUDIO 2017

- Installation process is divided into Workloads
  - Select “ASP.NET and web development” and “.NET Core cross-platform development”



CONFIRM THE INSTALL OF .NET CORE SDK

- Open Command Prompt
  - “where dotnet” => Installations for .NET Core
- Results for the following depend on the path
  - “dotnet” => “dotnet --info” =>
    - .NET Core CLI Info (2.0.0)
    - Shared Framework Host (2.0.0)
  - “dotnet --version” => .NET Core CLI Version Number (2.0.0)

CHANGE THE .NET CORE VERSION FOR A PROJECT

- Create a global.json file at the root

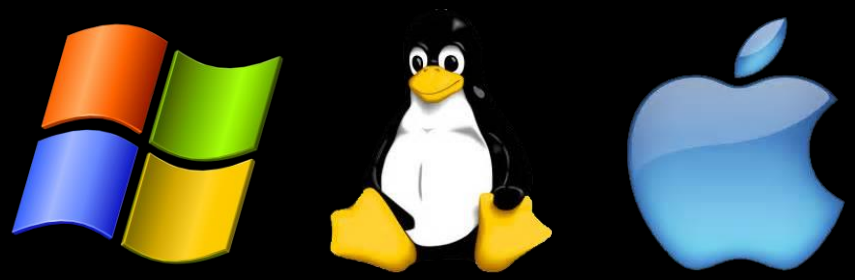
```
{
     "project" : ["src","test"],
     "sdk" : {
          "version" : "1.1.1"
     }
}
```

Lab 0: Installing Prereqs

INTRO TO .NET CORE 2.0

WHAT IS .NET CORE?

- Rewrite of “full” .NET Framework
- Vast performance improvements over prior versions
  - Including native compilation
- Flexible deployment/development model
  - Windows, Linux, Mac



- Full command line support
- True side by side installation support
- Open source from the start
  - Many improvements and features provided by the community

ANATOMY OF A .NET CORE APPLICATION (BOTTOM TO TOP)

- .NET Core Runtime (CoreCLR) - GC, JIT Compiler, base .NET Types
- .NET Core Framework Libraries (CoreFX) - Mostly platform/OS agnostic
- Application Host (dotnet.exe) and Command Line Interface (CLI)
- Custom Applications - Console Apps or Class libraries

FULL BCD (BIRTHDAY CAKE DIAGRAM)

.NET FRAMEWORK	.NET CORE	XAMARIN	TOOLS
WINDOWS APPLICATIONS	CROSS-PLATFORM SERVICES	MOBILE APPLICATIONS	 Visual Studio  Visual Studio Code  Xamarin Studio

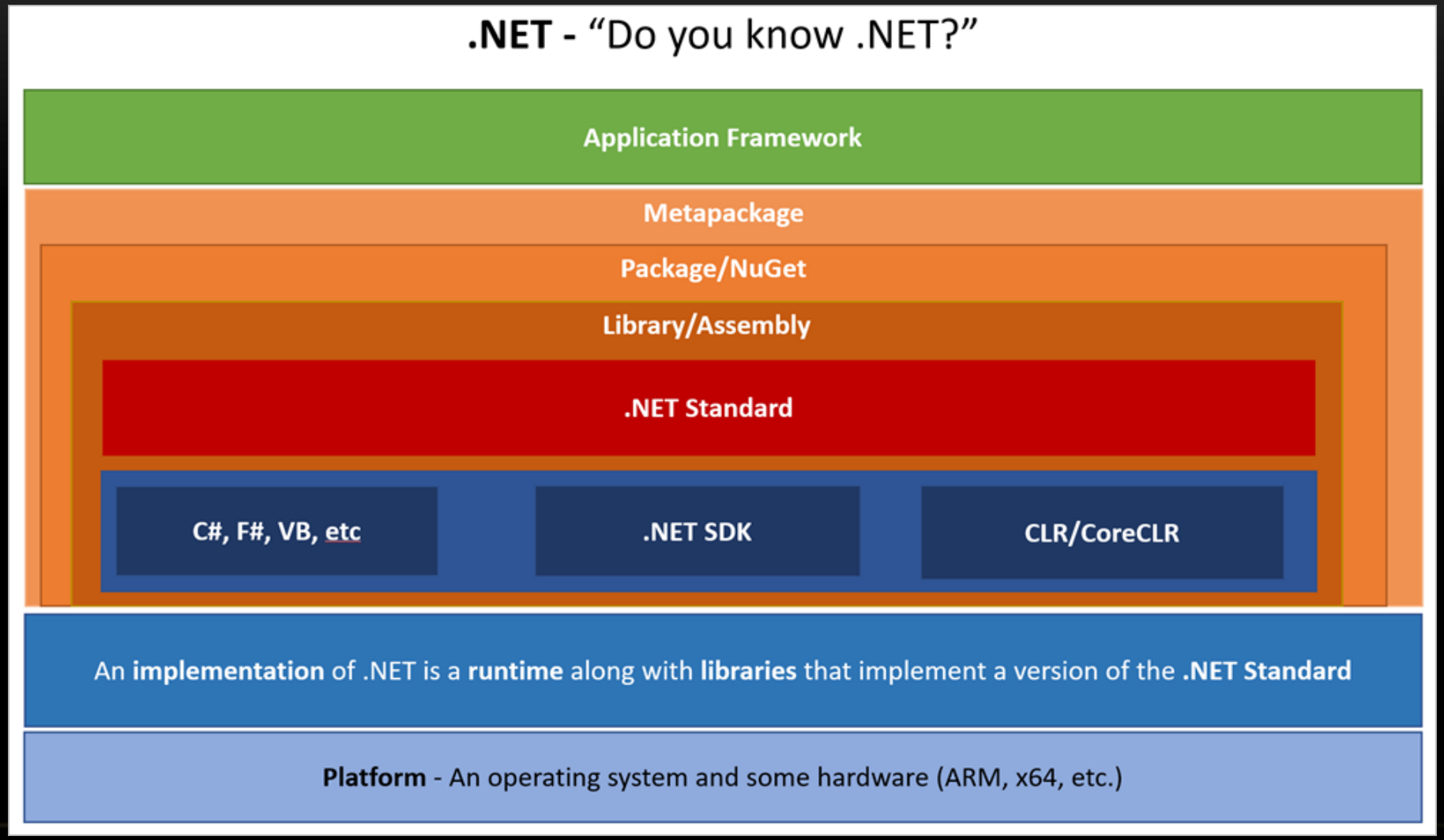
.NET STANDARD LIBRARY

COMMON INFRASTRUCTURE

Compilers | **Languages** | **Runtime components**

Courtesy of Rowan Miller
https://github.com/rowanmiller/Demo-EFCore

ZOOMING IN ON .NET



Courtesy of Scott Hanselman: https://www.hanselman.com/blog/DraftNETGlossaryDiagram.aspx

DEPLOYMENT

- Deployment models
  - Self contained –includes .NET Core f/w
  - Portable – expects .NET Core installed on deployment machine

- Kestrel adds a layer of complexity – see the docs

.NET CORE SUPPORT LIFECYCLES

- Long Term Support (LTS)
  - Major releases (e.g. **1**.0, **2**.0)
  - Only upgraded with critical fixes (patches)
  - Supported for three years after GA release or at least one year after the next LTS release.
  - NOTE: 1.1 was added to the LTS list with the release of 2.0
- Current
  - Minor releases (e.g. 1.**1**, 1.**2**)
  - Upgraded more rapidly
  - Supported for three months after next Current release

https://www.microsoft.com/net/core/support

WHAT'S NEW IN 2.0

- .NET Standard 2.0
  - Over 32K APIs (from 13K)
  - Also available in Azure Web Apps
- 6 new platforms supported
  - Can target Linux as "single" OS
- .NET Core SDK
  - .NET Core can reference .NET F/W Packages and Projects
  - "dotnet restore" is now implicit
- Performance Improvements
  - Profile-guided optimizations
  - Too many others to list…
- .NET Standard 2.0 NuGet Packages
  - F/W Dependencies removed
- Visual Basic support
  - Console apps, class libraries
- Live Unit Testing .NET Core 2
- Docker updates

.NET CORE APPLICATIONS ADDING/UPDATING PACKAGES

PROJECT CONSIDERATIONS

- MVC:
  - Select ‘ASP.NET Core Web Application (.NET Core)’
  - Choose ‘ASP.NET Core 2.0’ Templates
  - Select ‘Web Application (Model View Controller)’
- Data Access Library
  - Pick ‘Class Library (.NET Core)’
- Models
  - Pick ‘Class Library (.NET Core)’
- Unit Tests
  - Pick ‘xUnit Test Project (.NET Core)’

ADDING/UPDATING NUGET PACKAGES

- NuGet Packages update faster than VS2017 Templates
- Add/Update/Remove packages using:
  - .NET Core Command Line Interface
  - Package Manager Console
  - NuGet Package Manager GUI

RUNNING ASP.NET CORE APPLICATIONS

- Visual Studio
  - Select IIS or Kestrel
  - Port is controlled by launchSetting.json

- .NET Core CLI
  - ‘dotnet run’
  - Port defaults to 5000
    - Can be changed using WebHostBuilder

Lab 1:

Creating the Projects

Adding/Updating the NuGet packages

ENTITY FRAMEWORK CORE

EF PROJECT STATUS

WHAT IS ENTITY FRAMEWORK CORE

- Newest version of Entity Framework - complete re-write from EF 6.x
- Lightweight, Modularized
- Cross Platform (built on .NET Core)
- Based on an ‘Opt-in’ model – only load needed packages

- Just released EF Core 2.0
  - Many more features added
  - Still some missing features from EF 6.x
  - Check http://bit.ly/ef6_efcore to see the current status

(SOME) MISSING* FEATURES IN CURRENT VERSION OF EF CORE 2

- EDMX Designer
  - Not coming back!
- Alternate inheritance mapping patterns
  - Implemented: Table Per Hierarchy (TPH)
  - Missing: Table Per Type (TPT), Table Per Concrete Type (TPC)
- Spatial Data Types
- Lazy loading
- Command Interception
- Stored Procedure Mapping
- Data Initializers
- Some Data Annotations

http://bit.ly/ef6_efcore

FEATURES ADDED TO EF CORE OVER EF 6

- Batching of Statements (1.0)
- Shadow State Properties (1.0)
- Alternate Keys (1.0)
- Client side key generation (1.0)
- Field Mapping (1.1)
- Mixed Client/Server evaluation (1.0)
- Raw SQL with LINQ (1.0)
- DbContext Pooling (2.0)
- Like query operator (2.0)
- Global Query Filters (2.0)
- String interpolation with raw SQL (2.0)
- Scalar function mapping (2.0)
- Explicitly compiled LINQ queries (2.0)
- Attach a graph of new and existing entities (2.0)

EF CORE GOODNESS

DBCONTEXT

- EF Core DbContext changed since EF 6.x
  - Fully embraces dependency injection
- OnConfiguring provides fall back mechanism
- IDesignTimeDbContextFactory<TContext>
  - Assists with Context Pooling and Migrations
- Full support for FluentAPI in OnModelCreating
- Pooling support in ASP.NET Core 2.0

CONCURRENCY CHECKING (CARRY OVER)

- SQL Server uses Timestamp (rowversion) properties
  - Coded as a byte[] in C#
- Updates and Deletes are modified
  - Where <pk> = @p1 and <timestamp> = @p2
- Error throws DbUpdateConcurrencyException
  - Provides access to entities not persisted
- Developer decides how to handle concurrency errors

DBSET<T> FIND METHOD (RE-INTRODUCED IN 1.1)

- Introduced in EF Core 1.1
  - Largely due to the developer community
- Searches on primary key(s)
  - Returns instance from DbChangeTracker is currently tracked
  - Else calls to database

CONNECTION RESILIENCY (RE-INTRODUCED IN 1.1)

- Built in retry mechanism defined by relational database providers
  - Default – no retry
  - SqlServerRetryingExecutionStrategy
    - Optimized for SQL Server and SQL Azure
- Custom Execution Strategy
  - Specify retry count and max delay

- Throws RetryLimitExceededException
  - Actual exception is inner exception

USING COMPUTED COLUMNS IN MODELS (FIXED IN 1.1)

- Same table computed columns supported with EF Core 1.0

```
entity.Property(e => e.LineItemTotal).HasColumnType("money")
  .HasComputedColumnSql("[Quantity]*[UnitCost]");
```

- UDF based computed columns supported with EF Core 1.1

```
entity.Property(e => e.OrderTotal).HasColumnType("money")
  .HasComputedColumnSql("Store.GetOrderTotal([id])");
```

EF CORE MIGRATIONS (IMPROVED)

- Used to modify schema of based on model and SQL Code
  - Can also scaffold existing database into Context and Models
- Supports more than one DbContext in a project
  - E.g. ApplicationDbContext (ASP.NET Identity) and MyDomainModelContext
- Can also create SQL script representing changes to the database

CHANGES FROM EF6 MIGRATIONS

- The Good
  - No longer uses a hash to check database state
  - ModelSnapshot is C# file that contains all of the DDL
  - Database.Migrate method creates model AND runs all migrations

- The bad?
  - Database Initializers and Configuration Seed method are gone

PERFORMANCE IMPROVEMENTS (1.0)

- EF Core batches multiple insert, update, delete statements into a single call
  - Uses table valued parameters to process changes in a single network call
  - Improved performance through reduced network traffic
  - Reduces cost for cloud based databases
- Batch size can be configured through the DbContextOptions

SHADOW STATE PROPERTIES & ALTERNATE KEYS (1.0)

- Shadow State Properties
  - Properties not defined in your .NET classes
  - Maintained purely in the Change Tracker
  - Commonly used with foreign keys
- Alternate Keys
  - Non-primary keys used as foreign key fields
  - Conventions will create unique index for you

FIELD MAPPING/BACKING FIELDS (1.0)

- Allows EF to read and/or write to fields instead of properties
- Conventions
  - [m]_<camel-cased property name>
  - [m]_<property name>
- Fluent API
  - modelBuilder.Entity<Blog>()
    .Property(b=>b.Url).HasField("_theUrl")
- Used when materializing objects
  - Public getters/setters (if they exist) used at other times
- Can control when the fields are used
  - Field
  - FieldDuringConstruction
  - Property

EF CORE SUPPORTS MIXED EVALUATION (1.0)

- EF Core supports queries being evaluated on the server and the client
  - What executes where is provider specific
- Useful for including C# functions into the LINQ query/project
- Be careful where the client functions are injected
  - Poor usage can crush performance
- Enabled or disabled at the context level

```
optionsBuilder.UseSqlServer(connectionString)
  .ConfigureWarnings(warnings =>
        warnings.Throw(RelationalEventId.QueryClientEvaluationWarning));
```

POPULATING MODELS WITH RAW SQL QUERIES (1.0)

- Models can be populated from raw SQL using FromSql on DbSet<T>
  - Select list names must match the names that properties are mapped to
  - All fields on the model must be returned
- Useful for times when Sprocs or UDFs perform better than LINQ/EF
- Can also populate POCOs that are not tables
  - Must be in the Context as a DbSet<T>
  - Must have a primary key defined
- Can be mixed with LINQ statements

LIKE QUERY OPERATOR (2.0)

- Contained in the EF.Functions property
- Implemented at the database provider level
- You must add in the % yourself

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

//creates this query
SELECT [c].[Id], [c].[Name]
FROM [Customers] AS [c]
WHERE [c].[Name] LIKE N'a%';
```

GLOBAL QUERY FILTERS (2.0)

- Model level filters defined directly on the model
- Automatically applied to any queries on that type
  - Also applied to indirect queries (e.g. using Include or ThenInclude)
  - Can be used for soft deletes or multi-tenancy

```
public class BloggingContext : DbContext
{
    public DbSet<Post> Posts { get; set; }
    public int TenantId {get; set; }
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Post>().HasQueryFilter(
            p => !p.IsDeleted && p.TenantId == this.TenantId );
    }
}
```

STRING INTERPOLATION WITH RAW SQL QUERIES (2.0)

- Implemented in FromSql and ExecuteSqlCommand
- C# string interpolation items get converted into SQL parameters

```
var city = "London";
var contactTitle = "Sales Representative";
using (var context = CreateContext())
{
  context.Set<Customer>()
    .FromSql($@"SELECT * FROM ""Customers""
        WHERE ""City"" = {city} AND
           ""ContactTitle"" = {contactTitle}")
    .ToArray();
  }
```

```
@p0='London' (Size = 4000)
@p1='Sales Representative' (Size = 4000)

SELECT *
FROM ""Customers""
WHERE ""City"" = @p0
    AND ""ContactTitle"" = @p1
```

SCALAR FUNCTION MAPPING & EXPLICITLY COMPILED QUERIES (2.0)

- Scalar Function Mapping
  - Can map scalar functions to C# methods and used in LINQ queries
    - Use DbFunctionAttribute and make the method static on DbContext
    - Must create the function yourself
- Explicitly compiled queries
  - Create LINQ queries as static C# functions using EF.CompileQuery
  - Bypasses computation of hash and cache lookup

CHANGE TRACKING – ATTACH NEW AND EXISTING ENTITY GRAPH (2.0)

- Attach list of entities with DbContext.Attach or DbSet<T>.Attach
  - Entities with PK value will be marked as unchanged
  - Entities without PK value with be marked as added

Lab 2 Part 1:
Create the Models and ViewModels
Create the DbContext, Migrations
Add Calculated Column to Model (based on UDF)

FINISHING THE DATA ACCESS LAYER

CREATE THE REPOSITORIES

- DbContext is technically a combination of two patterns:
  - Unit of Work
  - Repository
- Adding Custom Repositories eliminates repetitive code
  - Create IRepo<T> and BaseRepo<T> to handle most scenarios
- Specific Repos (e.g. ProductsRepo) handle special cases

DATA INITIALIZATION

- Largely a manual process - Drop/Create base classes from EF 6.x don't exist
- EnsureDeleted() - Drops database
- EnsureCreated() - Creates database *based on model*
- Migrate() - Mutually exclusive of EnsureCreated()
  - Creates database and runs all migrations as well
- No way to set EF 6 style initializer – must call from code

Lab 2 Part 2:

Create the Repositories

Create the Data Initializer

TESTING EF CORE

XUNIT TEST FRAMEWORK

- Excellent .NET testing framework
- Built by the creators of nUnit
- Supports full .NET F/W, .NET Core, and Xamarin
  - Project templates are “in the box”
- Supports multitude of test runners
  - VS2017, R#, TestDriven.NET, TeamCity, MSBuild

XUNIT FUNDAMENTALS

- Fact = Test
- Theory = RowTest
- SetUp and TearDown removed in favor of constructors and IDisposable
- ExpectedException removed (Finally!)
- Full Generics support
- Use of anonymous delegates in Assert.Throws

```
Assert.Throws<ExceptionType>(()=>operation());
```

Lab 3:

Testing EF Core

ASP.NET CORE 2.0 FUNDAMENTALS

ASP.NET CORE 2.0

- ASP.NET Core 2.0 rebuilt on top of .NET Core 2.0
- Single, cross-platform framework for web, services, and microservices
  - WebApi + MVC + Web Pages + Razor Pages = ASP.NET Core
- Takes advantage of .NET Core performance
  - Includes a high performance web server (Kestrel) built on LibUV

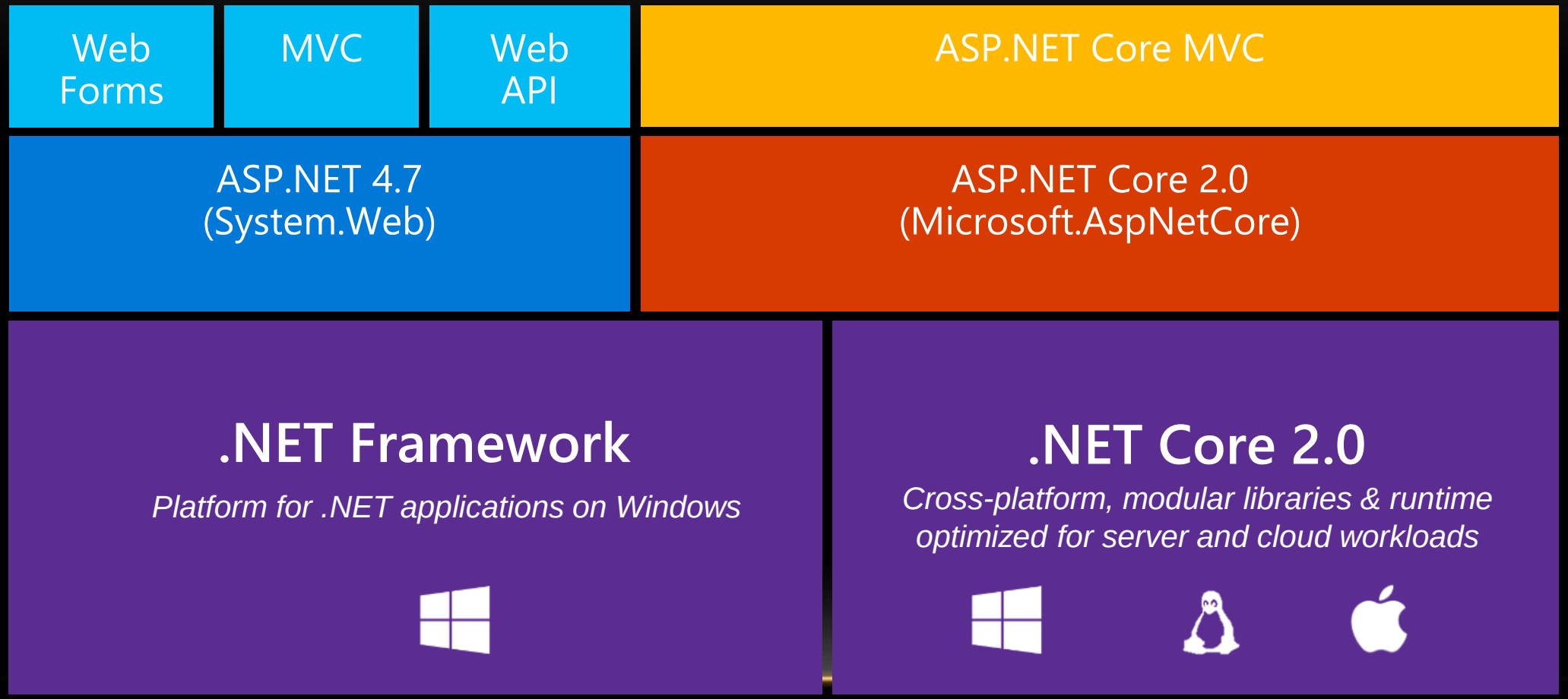
ASP.NET CORE FEATURES

- Pluggable Middleware
  - Routing, authentication, static files, etc.
- Full Dependency Injection integration
- Simplified and Improved Configuration System
- Tag Helpers
- View Components

WHAT'S NEW IN ASP.NET CORE 2.0

- Razor Pages
- Updated Templates
  - Razor pages, Angular, React
- DbContext Pooling with EF Core 2.0
- Razor support for C# 7.1
- Simplified configuration and startup
- Microsoft.AspNetCore.All metapackage
  - Includes EF SQL Server as well

ASP.NET CORE BCA



CONFIGURING THE WEB SERVER(S)

ASP.NET CORE APPS ARE CONSOLE APPS

- Web server(s) is(are) created in Program Main() method

```
var host = new WebHostBuilder()
                .UseKestrel()
                .UseContentRoot(Directory.GetCurrentDirectory())
                .UseIISIntegration()
                .UseStartup<Startup>()
                //Configuration will be discussed soon
                .ConfigureAppConfiguration(hostingContext,config)
                .UseUrls("http://*:40001/") //Configures Kestrel
                .Build();
host.Run();
```

- In ASP.NET Core 2.0, replaced with CreateDefaultBuilder()

LAUNCHSETTINGS.JSON CONTROLS RUNNING APP FROM VS

- IIS Settings
  - Sets app URL/SSL Port, auth settings
- Profiles (appear in VS Run command)
  - IIS Express
    - Sets environment variable
  - <AppName>
    - Sets URL, environment variable

APPLICATION CONFIGURATION

APPLICATION CONFIGURATION

- Simple JSON file configuration (by default)
  - appsettings.json
  - Other file types supported as well
- Environment determines files to load
  - appsettings.{environmentname}.json
  - Controlled by environment variable: ASPNETCORE_ENVIRONMENT
    - Built-in values of Development, Staging, Production

CUSTOM CONFIGURATION SECTIONS

- Add data to json

```
{
  "Logging": ...,
  "CustomSettings": {
    "ServiceAddress": "http://localhost:40002/",
    "ImageLocation": "~/images/"
  }
}
```

- Use IConfiguration to get information (in the DI container by default)

```
var customSection = Configuration?.GetSection("CustomSettings");
Var address = customSection?.GetSection("ServiceAddress")?.Value;
```

THE STARTUP CLASS

CONFIGURING THE PIPELINE

- The Configure method sets up how to respond to HTTP requests

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

CONDITIONAL PIPELINE CONFIGURATION

- Use environment options for conditional pipeline configuration

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env,
    ILoggerFactory loggerFactory)
{
        if (env.IsDevelopment())
        {
                app.UseDeveloperExceptionPage();
                app.UseBrowserLink();
        }
        else
        {
                app.UseExceptionHandler("/Home/Error");
        }
}
```

CONFIGURING FRAMEWORK SERVICES

- Used to configure any services needed by the application

```
public void ConfigureServices(IServiceCollection services)
{
   // Add framework services.
   services.AddMvc(config =>
   {
      config.Filters.Add(new SimpleAuthenticationActionFilter());
   })
   .AddJsonOptions(options =>
   { //Revert to PascalCasing for JSON handling
      options.SerializerSettings.ContractResolver = new DefaultContractResolver();
   });
   //Additional services for DI added here (covered later in this presentation)
}
```

CONFIGURING EF CORE CONTEXT POOLING

- New feature in ASP.NET/EF Core 2
- Context must have single public constructor that takes DbContextOptions

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContextPool<StoreContext>(options =>
       options.UseSqlServer(Configuration.GetConnectionString("SpyStore")));
}
```

DEPENDENCY INJECTION

ADDING CUSTOM DEPENDENCIES TO DI CONTAINER

- Configured in Startup.cs
- Used to configure any services needed by the application
  - Transient – created each time they are requested
  - Scoped – created once per request
  - Singleton – created once (use this instead of implementing singleton dp)

```
public void ConfigureServices(IServiceCollection services)
{
   services.AddSingleton<ICustomSettings>(new CustomSettings(Configuration));
   services.AddScoped<ICategoryRepo, CategoryRepo>();
}
```

INJECTING SERVICES INTO CONTROLLERS AND VIEWS

- Dependencies are configured in Startup.cs

```
public void ConfigureServices(IServiceCollection services)
{
    //https://docs.asp.net/en/latest/fundamentals/dependency-injection.html
    services.AddScoped<IShoppingCartRepo, ShoppingCartRepo>();
}
```

- Instances are pulled into classes automagically

```
public ShoppingCartController(IShoppingCartRepo repo)
{ //Omitted for brevity }
```

- Instances are pulled into views with the @inject directive

```
@inject <type> <name>
@inject IWebApiCalls apiCalls
```

Lab 4:
Creating the WebHost
Configuring the application
Adding connection strings to the settings files

CUSTOM VALIDATION

VALIDATION

- Nothing new for standard model validation
  - Validation and Display attributes
  - Model State
  - Explicit and Implicit validation
- Creating custom validation attributes changed slightly
  - Server side code derives from ValidationAttribute
    - Must also implement IClientModelValidator to support client side scripts
  - Client side validation ties into Jquery validations

SERVER SIDE VALIDATION

- Override ValidationResult IsValid method
  - ValidationContext provides access to metadata and the rest of the model
  - Return ValidationResult.Success or ValidationResult(errorMessage)
- Should also override FormatErrorMessage

CLIENT SIDE VALIDATION

- Must implement AddValidation method in custom attribute
  - Adds the data-val attributes to the rendered element only if using razor editor templates
- JavaScript code needs to:
  - Add validator method – must match data-val name
  - Add unobtrusive validation adapter
    - Must match data-val name
    - Rules must be set to enable validation
- Jquery.Unbotrusive-ajax.js must be referenced on the page

BUNDLING AND MINIFICATION

BUNDLING AND MINIFICATION

- JavaScript and CSS files should be bundled and minified for performance
- VS 2017 uses BundlerMinifer NuGet package by default
- Settings defined in by bundleconfig.json
  - Specify: outputfilename, inputfiles (globbing allowed), optional parameters
- Add:
  - BundlerMinifier Visual Studio Extension for IDE integration
  - BundlerMinifier.Core for .NET Core CLI

Lab 5:

Custom server and client side validation

Bundling and Minification

VIEW MODELS AND CONTROLLERS

CONTROLLERS

- Everything derives from Controller base class
  - Base class provides many helpers, such as:
  - Ok (200), BadRequest (400), NotFound (404)
- Actions return an IActionResult/Task<IActionResult>
- Dependencies are injected into the controllers
- Attribute Routing is now a first class citizen in ASP.NET Core
  - Helps to refine routing for individual controller actions
- In this example app, base controller OnActionExecuting override is used to create fake authentication

ROUTING

ROUTING

- Attribute Routing is first class citizen in ASP.NET Core
  - Helps to refine routing for individual controller actions
- Route table used for default route
  - Sometimes skipped to ASP.NET Core Service Applications
- Controller and actions can define specific routes

```
[Route("api/[controller]/{customerId}")]
public class ShoppingCartController : Controller
{
    [HttpGet("{Id?}")]
    public IActionResult AddToCart(int Id, int customerId, int quantity = 1)
    {
        //Code omitted
    }
}
```

Lab 6:

View Models and Controllers

TAG HELPERS AND VIEWS

TAG HELPERS

- Encapsulate server side code to shape the attached element
  - Keep developers “in the HTML”
- Most Razor HTML Helpers have corresponding Tag Helpers
  - Form, Anchor, Input, TextArea, Select, Validation, Link/Script, Image
  - Added as attributes with asp-
- Special: Environment Tag Helper
- Custom Tag Helpers can be created

TAG HELPER DETAILS

- Form
  - Similar to BeginForm/EndForm HTML Helper
  - Automatically generates the anti-forgery token
  - asp-controller, asp-action, asp-method, asp-route-<parameter name>
  - Can use named routes: asp-route="routename"
- Anchor
  - Similar to ActionLink HTML Helper
  - asp-controller, asp-action, asp-route-<parameter name>

TAG HELPER DETAILS (CONTINUED)

- Input
  - Model property is selected with asp-for (strongly typed)
  - Generates id and name properties for each element
  - Renders markup based on data type of the model property
  - Adds in HTML type based on model type and data annotations
  - Generates HTML5 validation attributes
- TextArea
  - Model property is selected with asp-for (strongly typed)
  - Generates id and name properties for each element

TAG HELPER DETAILS (CONTINUED)

- Select
  - Generates id and name properties for each element
  - Generates select and options using asp-for and asp-items
- Validation
  - Model validation uses asp-validation-summary (in a div)
    - Options are All, ModelOnly, None
  - Property validation uses asp-validation-for (in a span)
- Link, Image
  - Can append (asp-append-version="true") hash of the file to the URL to prevent caching issues

TAG HELPER DETAILS (CONTINUED)

- Script
  - Provide fallback for scripts (such as CDN sources)
    - Fallback source – asp-fallback-src
    - Test for fallback – asp-fallback-test
- Environment
  - Tag helper to define markup block based on configuration environment

```
<environment include="Development“ exclude="Staging,Production">
```

Lab 7:

Views

VIEW COMPONENTS

VIEW COMPONENTS

- Combine partial views with server side capabilities
- Server side class implements ViewComponent
- Partial view must be located in:
  - Views/<controller_name>/Components/<view_component_name>/<view_name>
  - Views/Shared/Components/<view_component_name>/<view_name>
- Don’t use model binding
- Can be invoked as a Tag Helper (with ASP.NET Core 1.1)

Lab 8:

View Components

CUSTOM TAG HELPERS

CUSTOM TAG HELPERS

- Composed entirely of server side code
- Class inherits TagHelper
- Class name (minus TagHelper) becomes the element name
  - E.g. EmailTagHelper == <email></email>
- Public properties are added as lower kebob cased attributes
  - E.g. EmailName == email-name=“”
- Must opt in to use (usually in the _ViewImports.cshtml partial)
  - @addTagHelper *, SpyStore_HOL.MVC

Lab 9:

Custom Tag Helpers

Contact Me

skimedic@outlook.com
www.skimedic.com/blog
www.twitter.com/skimedic

http://bit.ly/skimediclyndacourses
http://bit.ly/apressbooks

www.hallwayconversations.com

Thank You!